

Efficient Modelling and Simulation Methodology for the Design of Heterogeneous Mixed-Signal Systems on Chip

THÈSE N° 4993 (2011)

PRÉSENTÉE LE 15 AVRIL 2011

À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE SYSTÈMES MICROÉLECTRONIQUES
PROGRAMME DOCTORAL EN MICROSYSTÈMES ET MICROÉLECTRONIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Torsten MÄHNE

acceptée sur proposition du jury:

Prof. M. A. Ionescu, président du jury
Prof. Y. Leblebici, A. Vachoux, directeurs de thèse
Prof. D. Atienza Alonso, rapporteur
Prof. A. Jantsch, rapporteur
Prof. I. O'Connor, rapporteur



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

Résumé

Les systèmes intégrés sur puce ou en boîtier sont des composants clés de gammes de produits très variés, qui vont des cartes à puces en passant par les téléphones mobiles aux voitures. En plus de composants logiques/numériques de plus en plus complexes et logiciels embarqués pour le traitement et stockage de données, ils peuvent intégrer des composants analogiques/RF, des capteurs et actuateurs pour interagir avec leur environnement analogique. Cette tendance vers des systèmes plus complexes et hétérogènes supportant plus des fonctionnalités entremêlées est rendu possible par les avancées continues des technologies de fabrication et de production et accéléré par la demande pour des nouveaux produits. C'est pourquoi la réutilisation et la migration de composant existants deviennent de plus en plus importants. Tous ces facteurs rendent le processus de conception de plus en plus complexe et multidisciplinaire. Aujourd'hui, la conception des composants individuels est bien maîtrisée et optimisée par l'utilisation d'une grande diversité de logiciels CAO/EDA, de langages de conception et de formats de données. Ils sont basés sur des concepts de modélisation/abstraction, des formalismes de descriptions (aussi appelés *modèles de calcul*) et des méthodes d'analyse/simulation spécifiques. Le concepteur doit combler le vide entre les outils et méthodologies en utilisant des conversions manuelles de modèles et des liaisons/intégrations propriétaires des outils de conception/simulation, un travail complexe et sujet à erreurs. Il manque ainsi une méthodologie de conception commune et une plateforme indépendante pour la gestion, l'échange et le développement collaboratif des modèles de différents formats et différents niveaux d'abstraction. La vérification du système complet est un grand problème, qui nécessite l'existence de modèles compatibles pour chaque composant et au bon niveau d'abstraction pour achever des résultats satisfaisants par rapport à la fonctionnalité du système et à la couverture des tests, tout en gardant une performance de simulation satisfaisante en termes de vitesse et précision. Ainsi, le grand défi est l'intégration parallèle de tous ces divers sous-processus de conception. Les concepteurs ont donc besoin d'une plateforme de conception et simulation commune pour créer et raffiner une spécification exécutable du système complet (un prototype virtuel) à un haut niveau d'abstraction, qui supporte plusieurs modèles de calcul. Ceci permet l'exploration des différentes options architecturales, l'estimation des performances du système, la validation des composants réutilisés, la vérification des interfaces entre des composants hétérogènes et l'interopérabilité avec des autres systèmes ainsi que le bilan des influences provenant de l'environnement d'opération et des tolérances de fabrication utilisées pour la création de ces systèmes. Pour les systèmes embarqués analogiques et mixtes, la plateforme SystemC et ses extensions AMS, à la standardisation desquelles l'auteur a contribué, sont en train de s'établir.

Cette thèse décrit la contribution de l'auteur à la résolution des problèmes de modélisation et simulation susmentionnés dans trois phases thématiques. Dans la première phase, le prototype d'une plateforme basée sur le web, appelée *ModelLib*, pour collecter des modèles de domaines et niveaux d'abstractions différents et leur méta-informations structurelles et sémantiques a été développée. Ce travail inclut l'implémentation d'un mécanisme de contrôle d'accès hiérarchique, qui permet la protection de la propriété intellectuelle constituée par les modèles à des différents niveaux de détail. Des cas d'utilisation ont été développés, qui montrent comment l'outil peut supporter le processus de conception de systèmes sur puce analogiques/mixtes en renforçant la réutilisation et le développement commun des modèles pour des tâches comme l'exploration architecturale, la validation du système et la création progressive de modèles plus élaborés du système.

Les expériences du développement de ModelLib ont permis d'identifier quels aspects doivent être adressés particulièrement pendant le développement des modèles pour les rendre réutilisables : notamment flexibilité, documentation et validation. Ceci a constitué le point de départ pour le développement d'une méthodologie de modélisation efficace pour la conception descendante (top-down) et la vérification montante (bottom-up) de systèmes RF basées sur l'utilisation systématique des modèles comportementaux. Un résultat est le développement d'une bibliothèque de modèles VHDL-AMS bien documentés, paramétrables et fidèles au niveau circuit des composants analogiques/numériques/RF typiques d'un émetteur-récepteur. Les modèles proposent deux gammes de paramètres au concepteur : une qui est basée sur les spécifications de performances et une qui est basée sur les paramètres des composants du circuit, repris de l'implémentation au niveau des transistors. Le niveau d'abstraction pour la description du comportement analogique/numérique/RF des composants a été choisi pour atteindre un bon compromis entre précision, fidélité et performance de simulation. Les interfaces permettent l'intégration des modèles au niveau de transistor pour la validation de modèles comportementaux et la vérification de l'implémentation des composants dans le contexte du système complet. Ces propriétés rendent les modèles utilisables pour des différentes tâches de conception comme l'exploration architecturale ou la validation du système. Ceci est démontré sur un modèle d'un émetteur-récepteur à modulation par déplacement de fréquence binaire (BFSK), qui peut être paramétré pour satisfaire des spécifications très différentes. Ce projet démontre aussi les limitations des langages de description de matériel classiques en terme d'abstraction et de performance de simulation.

C'est pourquoi la troisième et dernière phase a été dédiée à augmenter le niveau d'abstraction pour la description des systèmes sur puce analogiques et mixtes hétérogènes et complexes, afin de permettre leur simulation efficace en utilisant des différents modèles de calculs synchronisés. Ce travail utilise la plateforme de simulation SystemC basée sur C++ et ses extensions AMS. Des nouvelles capacités de modélisation, qui vont au-delà des extensions AMS standard pour SystemC, ont été développées pour décrire des systèmes multi-domaines à conservation d'énergie dans une façon formelle et cohérente à un haut niveau d'abstraction. Dans ce but, toutes les constantes, variables et paramètres d'un modèle de système qui représentent une quantité physique peuvent déclarer leur dimension et système d'unités comme une partie intrinsèque du type de donnée. Leurs affectations doivent ainsi spécifier une valeur et son unité de mesure. Cela permet une définition bien plus précise mais toujours compacte des interfaces et équations du modèle. Cela permet aussi au compilateur C++ de vérifier l'assemblage correct des composants et la cohérence des équations par analyse dimensionnelle statique. L'implémentation est basée sur la bibliothèque Boost.Units, qui utilise des techniques de métaprogrammation avec des patrons. Un filtre dédié pour les types de données représentant les unités de mesure a été développé pour simplifier les messages du compilateur et faciliter la recherche des erreurs d'unité. Pour assurer que les modèles soient réutilisables malgré leur interface très spécifique, leur interfaces et comportements doivent être paramétrables d'une façon bien définie. Les techniques d'implémentation nécessaires ont été démontrées par le développement d'une bibliothèque de modèles génériques des composants de schémas des blocs pour le modèle de calcul Timed Data Flow (TDF) des extensions AMS de SystemC. Ces techniques sont aussi la clé pour l'intégration d'un nouveau modèle de calcul basé sur le formalisme de graphe de liaisons (bond graphs) dans les extensions AMS de SystemC. Les graphes de liaisons facilitent la description unifiée des parties à conservation d'énergie de systèmes hétérogènes à l'aide d'un ensemble limité de primitives de modélisation paramétrables au domaine physique et permettent leur simulation avec des performances comparables à celles d'un modèle de flot de signaux équivalent.

Mots-clés : analyse dimensionnelle ; graphe de liaisons (bond graph) ; bibliothèque de modèles ; méthodologie de modélisation ; simulation ; modèle de calcul ; systèmes sur puce analogique/mixtes/RF et multi-physique ; OSCI SystemC AMS extensions ; VHDL-AMS.

Zusammenfassung

Elektronische Einchip- und Eingehäusesysteme sind Schlüsselkomponenten einer immer größeren Vielfalt an Produkten, angefangen von Chipkarten über Mobiltelefone bis hin zu Kraftfahrzeugen. Neben zunehmend komplexerer digitaler Hard- und Software zur Datenverarbeitung und -speicherung integrieren sie immer mehr analoge/RF Schaltungen, Sensoren und Aktoren, um mit ihrer (analogen) Umgebung zu interagieren. Dieser Trend, hin zu immer komplexeren und heterogeneren Systemen mit mehr ineinander greifenden Funktionen, wird durch die anhaltenden Fortschritte in der Fertigungstechnologie ermöglicht und durch die stetige Marktnachfrage nach neuen Produkten und Varianten getrieben. Die Wiederverwendung und Neuanpassung von bestehenden Komponentenentwürfen wird daher immer wichtiger. Durch all diese Faktoren wird aber auch der Entwurfsprozess immer komplexer und multidisziplinärer. Dabei ist der Entwurf der einzelnen Komponenten durch die betroffenen Fachdisziplinen gut verstanden und optimiert durch Einsatz verschiedenster Entwurfsautomatisierungsprogramme, Beschreibungssprachen und Datenformate. Diese basieren auf spezifischen Modellierungs-/Abstraktionskonzepten, Beschreibungssprachen (auch Berechnungsmodelle genannt) und Analyse-/Simulationsmethoden. Die Designer müssen die bestehenden Lücken zwischen den Werkzeugen und Entwurfsmethoden durch manuelle Übertragung der Modelle sowie durch selbstentwickelte Werkzeugkopplungen und -integration überbrücken, was fehleranfällig und zeitaufwändig ist. Es fehlt eine gemeinsame Entwurfsmethodik und eine unabhängige Plattform zur Verwaltung, dem Austausch und der gemeinsamen Entwicklung von Modellen verschiedener Formate und Abstraktionsniveaus. Die Verifikation des Gesamtsystems durch Simulation bezüglich seiner Funktionalität und mit guter Testabdeckung ist ein großes Problem, da sie kompatible Komponentenmodelle auf einem passenden Abstraktionsniveau voraussetzt, die eine zufriedenstellende Simulationsgeschwindigkeit und -präzision ermöglichen. Die Herausforderung besteht daher in der parallelen Integration dieser sehr unterschiedlichen Teilentwurfsprozesse. Die Designer benötigen dazu eine gemeinsame Entwurfs- und Simulationsplattform zur Erstellung und Verfeinerung einer ausführbaren Spezifikation des Gesamtsystems (virtueller Prototyp) auf einem hohen Abstraktionsniveau, das verschiedene Berechnungsmodelle unterstützt. Diese ermöglicht die Evaluierung von Architekturoptionen, die Abschätzung der Systemleistung, die Validierung wiederverwendeter Komponenten, die Verifikation der Schnittstellen zwischen den heterogenen Komponenten und der Interoperabilität mit anderen Systemen sowie die Abschätzung des Einflusses der zukünftigen Arbeitsumgebung und des genutzten Fertigungsprozesses. Für digital/analoge Hard-/Softwaresysteme ist das C++-basierte SystemC mit seinen AMS-Erweiterungen, zu deren Standardisierung der Autor beitrug, dabei sich als solche Plattform zu etablieren.

Diese Doktorarbeit beschreibt den Beitrag des Autors zur Lösung der genannten Modellierungs- und Simulationsprobleme in drei thematischen Phasen. In der ersten Phase wurde der Prototyp, genannt ModellLib, einer webbasierten Plattform zum Sammeln von Modellen verschiedener Disziplinen und Abstraktionsniveaus und den dazugehörigen semantischen und strukturellen Metainformationen entwickelt. Diese Arbeit beinhaltete auch die Implementierung eines Zugangskontrollmechanismus zum abgestuften Schutz des aus den Modellen gebildeten geistigen Eigentums. Die entwickelten Anwendungsfälle für dieses Werkzeug zeigen, wie es den Entwurfsprozess von Einchipsystemen unterstützen kann, indem es die Wiederverwendung und gemeinsame Entwicklung von Modellen erleichtert, für Aufgaben wie die Architekturauswahl, Systemvalidierung und Erstellung immer komplexerer Systemmodelle.

Die Erfahrungen aus der ModellLib-Entwicklung zeigten, welche Aspekte bei der Entwicklung wiederverwendbarer Modelle besonders beachtet werden müssen: Flexibilität, Dokumentation und Validierung. Dies war in der zweiten Phase der Ausgangspunkt zur Entwicklung einer effizienten Modellierungsmethodologie für die Top-Down-Entwicklung und Bottom-Up-Verifikation von RF-Systemen basierend auf dem systematischen Einsatz von Verhaltensmodellen. Ein Ergebnis ist die entwickelte Bibliothek gut dokumentierter, flexibel parametrierbarer und pinakkurater VHDL-AMS-Modelle von typischen analogen, digitalen und RF-Komponenten eines Sender/Empfängers. Die Komponentenmodelle bieten dem Entwickler zwei Parametersätze an: einer basiert auf der Leistungsspezifikation und der andere basiert auf den aus der Schaltungsebene zurückübernommenen Bauteilparametern. Dabei wurde das Abstraktionsniveau der Modelle so gewählt, dass sie einen attraktiven Ausgleich zwischen den Anforderungen Präzision, vielseitige Verwendbarkeit und hoher Simulationsgeschwindigkeit erzielen. Die pinakkuraten Modellschnittstellen erlauben eine einfache Integration von Modellen von der Transistorebene zur Validierung der Verhaltensmodelle oder der Verifizierung der Komponentenimplementierung im Kontext des Gesamtsystems. Diese Eigenschaften machen die Modelle, von der Architekturauswahl bis hin zur Validierung des Gesamtsystems, vielseitig einsetzbar. Demonstriert wurde dies am Modell eines Senders mit zweiwertiger Frequenzumtastung (BFSK), welches durch seine Parameter an sehr unterschiedliche Zielspezifikationen anpassbar ist. Diese Arbeit zeigte auch die Grenzen klassischer Hardwarebeschreibungssprachen bezüglich erreichbaren Abstraktionsniveaus und Simulationsgeschwindigkeit auf.

Das Ziel der dritten und letzten Phase war es daher das Abstraktionsniveau bei der Beschreibung komplexer, heterogener Systeme anzuheben und durch den Einsatz verschiedener synchronisierter Berechnungsmodelle ihre effiziente Simulation zu ermöglichen. Diese Arbeit setzt auf den OSCI SystemC AMS extensions auf. Neue Modellierungsfähigkeiten, die über ihren standardisierten Umfang hinausgehen, wurden eingeführt, um energieerhaltende, multiphysikalische Systeme formal und konsistent beschreiben zu können. Alle Konstanten, Variablen und Parameter eines Systemmodells, die physikalische Größen repräsentieren, können nun ihre Dimension und das zugehörige Einheitensystem als intrinsischen Teil ihres Datentyps deklarieren. Zuweisungen an diese müssen dann neben dem Wert auch die korrekte Maßeinheit enthalten. Dies ermöglicht für die Modelle eine viel präzisere und dennoch kompakte Definition der Schnittstellen und Gleichungen, so dass nun der C++-Compiler die korrekte Verdrahtung der Komponenten und die Konsistenz der Gleichungen durch Dimensionsanalyse prüfen kann. Die Implementierung basiert auf der Boost.Units-Bibliothek, die dafür Template-Metaprogrammierungstechniken nutzt. Ein speziell implementierter Filter für die Maßeinheitendatentypen in den Compilermeldungen vereinfacht die Suche nach Einheitenfehlern. Um trotz präzis definierter Schnittstellen die Wiederverwendung von Modellen sicherzustellen, müssen ihre Schnittstellen und Verhalten auf wohldefinierte Weise parametrierbar sein. Die dazu notwendigen Implementierungstechniken demonstriert die entwickelte generische Bibliothek von Modellen typischer Blockdiagrammelemente für das Timed Data Flow (TDF)-Berechnungsmodell. Diese Techniken sind auch die Basis für die Implementierung eines neuen Berechnungsmodells für die SystemC AMS extensions zur Unterstützung des Bondgraphenformalismus. Dieser ermöglicht eine vereinheitlichte Beschreibung der energieerhaltenden Teile eines heterogenen Systems mit Hilfe eines kleinen Satzes von, auf die Energiedomäne parametrierbaren, Elementen und ihre Simulation mit einer hohen Geschwindigkeit vergleichbar mit der eines äquivalenten Signalfussplanmodells.

Schlagwörter: Dimensionsanalyse; Bondgraph; Modellbibliothek; Modellierungsmethodologie; Simulation; Berechnungsmodell; multidisziplinäre RF/analog/digitale Einchipssystem; OSCI SystemC AMS extensions; VHDL-AMS.

Abstract

Systems on Chip (SoCs) and Systems in Package (SiPs) are key parts of a continuously broadening range of products, from chip cards and mobile phones to cars. Besides an increasing amount of digital hardware and software for data processing and storage, they integrate more and more analogue/RF circuits, sensors, and actuators to interact with their (analogue) environment. This trend towards more complex and heterogeneous systems with more intertwined functionalities is made possible by the continuous advances in the manufacturing technologies and pushed by market demand for new products and product variants. Therefore, the reuse and retargeting of existing component designs becomes more and more important. However, all these factors make the design process increasingly complex and multidisciplinary. Nowadays, the design of the individual components is usually well understood and optimised through the usage of a diversity of CAD/EDA tools, design languages, and data formats. These are based on applying specific modelling/abstraction concepts, description formalisms (also called *Models of Computation* (MoCs)) and analysis/simulation methods. The designer has to bridge the gaps between tools and methodologies using manual conversion of models and proprietary tool couplings/integrations, which is error-prone and time-consuming. A common design methodology and platform to manage, exchange, and collaboratively develop models of different formats and of different levels of abstraction is missing. The verification of the overall system is a big problem, as it requires the availability of compatible models for each component at the right level of abstraction to achieve satisfying results with respect to the system functionality and test coverage, but at the same time acceptable simulation performance in terms of accuracy and speed. Thus, the big challenge is the parallel integration of these very different part design processes. Therefore, the designers need a common design and simulation platform to create and refine an executable specification of the overall system (a virtual prototype) on a high level of abstraction, which supports different MoCs. This makes possible the exploration of different architecture options, estimation of the performance, validation of re-used parts, verification of the interfaces between heterogeneous components and interoperability with other systems as well as the assessment of the impacts of the future working environment and the manufacturing technologies used to realise the system. For embedded Analogue and Mixed-Signal (AMS) systems, the C++-based SystemC with its AMS extensions, to which recent standardisation the author contributed, is currently establishing itself as such a platform.

This thesis describes the author's contribution to solve the modelling and simulation challenges mentioned above in three thematic phases. In the first phase, the prototype of a web-based platform to collect models from different domains and levels of abstraction together with their associated structural and semantical meta information has been developed and is called *ModelLib*. This work included the implementation of a hierarchical access control mechanism, which is able to protect the Intellectual Property (IP) constituted by the model at different levels of detail. The use cases developed for this tool show how it can support the AMS SoC design process by fostering the reuse and collaborative development of models for tasks like architecture exploration, system validation, and creation of more and more elaborated models of the system.

The experiences from the ModelLib development delivered insight into which aspects need to be especially addressed throughout the development of models to make them reusable: mainly flexibility, documentation, and validation. This was the starting point for the development of an efficient modelling

methodology for the top-down design and bottom-up verification of RF Systems based on the systematic usage of behavioural models in the second phase. One outcome is the developed library of well documented, parameterisable, and pin-accurate VHDL-AMS models of typical analogue/digital/RF components of a transceiver. The models offer the designer two sets of parameters: one based on the performance specifications and one based on the device parameters back-annotated from the transistor-level implementation. The abstraction level used for the description of the respective analogue/digital/RF component behaviour has been chosen to achieve a good trade-off between accuracy, fidelity, and simulation performance. The pin-accurate model interfaces facilitate the integration of transistor-level models for the validation of the behavioural models or the verification of a component implementation in the system context. These properties make the models suitable for different design tasks such as architecture exploration or overall system validation. This is demonstrated on a model of a binary Frequency-Shift Keying (FSK) transmitter parameterised to meet very different target specifications. This project showed also the limits in terms of abstraction and simulation performance of the “classical” AMS Hardware Description Languages (HDLs).

Therefore, the third and last phase was dedicated to further raise the abstraction level for the description of complex and heterogeneous AMS SoCs and thus enable their efficient simulation using different synchronised MoCs. This work uses the C++-based simulation framework SystemC with its AMS extensions. New modelling capabilities going beyond the standardised SystemC AMS extensions have been introduced to describe energy conserving multi-domain systems in a formal and consistent way at a high level of abstraction. To this end, all constants, variables, and parameters of the system model, which represent a physical quantity, can now declare their dimension and associated system of units as an intrinsic part of their data type. Assignments to them need to contain besides the value also the correct measurement unit. This allows a much more precise but still compact definition of the models’ interfaces and equations. Thus, the C++ compiler can check the correct assembly of the components and the coherency of the equations by means of dimensional analysis. The implementation is based on the Boost.Units library, which employs template metaprogramming techniques. A dedicated filter for the measurement units data types has been implemented to simplify the compiler messages and thus facilitate the localisation of unit errors. To ensure the reusability of models despite precisely defined interfaces, their interfaces and behaviours need to be parametrisable in a well-defined manner. The enabling implementation techniques for this have been demonstrated with the developed library of generic block diagram component models for the Timed Data Flow (TDF) MoC of the SystemC AMS extensions. These techniques are also the key to integrate a new MoC based on the bond graph formalism into the SystemC AMS extensions. Bond graphs facilitate the unified description of the energy conserving parts of heterogeneous systems with the help of a small set of modelling primitives parametrisable to the physical domain. The resulting models have a simulation performance comparable to an equivalent signal flow model.

Keywords: dimensional analysis; bond graphs; model library; modeling methodology; simulation; Model of Computation (MoC); multiphysical, RF, and Analogue and Mixed-Signal (AMS)-Systems on Chip (SoCs); OSCI SystemC AMS extensions; VHDL-AMS.

Acknowledgements

This thesis marks not only the end of a six year long research project but also of a period of my life, which inspired and marked me in many ways. I will fondly remember all the people and events linked to it. Therefore, I would like to take the opportunity to thank some of the people without whom I wouldn't have come so far.

First of all, I would like to thank my advisor, Prof. Yusuf Leblebici, for giving me the opportunity to pursue this research in his lab. He gave me his confidence as well as provided me with motivation and support during all these years.

I am particular grateful to my co-advisor Dr Alain Vachoux for his continuous guidance, support, motivation, close collaboration, and continuous exposure to French throughout this research work. His door has been always open for me. In many fruitful technical discussions, he shared his experience with me and thus provided me with knowledge, insight, inspiration, and perspective, which significantly contributed to my scientific work. Working for him as a teaching assistant in his various courses was a demanding, valuable, and satisfying experience for me. Moreover, he provided me with great assistance during the course of writing this Ph.D. thesis and previous publications. It has been a great pleasure and honour to work with him so closely.

At the same time, I would like to thank the members of my thesis committee, Prof. Adrian Ionescu, Prof. David Atienza Alonso, Prof. Axel Jantsch, and Prof. Ian O'Connor for investing their time to read my thesis and evaluate my research work.

I would like to thank the members of the OSCI AMS working group—among them in particular Martin Barnasconi, Christoph Grimm, Karsten Einwich, Alain Vachoux, François Pêcheux, Marie-Minerve Louërat, Serge Scotti, Thomas Uhle, Markus Damm, Sumit Adhikari, and Philipp A. Hartmann—for the many fruitful technical discussions during all these years of standardisation of the SystemC AMS extensions, which formed the base of my own research work and significantly influenced it. It has been a unique experience to participate in it. In particular, I would like to thank Karsten Einwich and Thomas Uhle as the main developers of the SystemC-AMS proof-of-concept implementation at Fraunhofer IIS/EAS in Dresden, Germany, who gave me insight into the internals of SystemC-AMS and showed me how my ideas for a new model of computation based on the bond graph formalism could be integrated with their work during a six week stay at their institute. I am very thankful to Martin Barnasconi, who, despite his demanding work at NXP and continuous efforts as the chair of the AMS working group, found the time to carefully review my Ph.D. thesis and provide me with many valuable comments and corrections. I would also like to thank the authors of the OSCI SystemC proof-of-concept library implementation and the countless ones of the Boost C++ libraries—these open source libraries formed together with Fraunhofer's SystemC-AMS library the solid foundation for my own work in this domain.

I would like to thank the members of the RF group (secteur 161) of the Centre Suisse d'Électronique et de Microtechnique SA (CSEM) in Neuchâtel for the fruitful collaboration on the modelling methodology project for RF systems: Frédéric Giroud, Matteo Contaldo, Paola Tortori, and Vincent Peiris. Especially Frédéric's experience as an RF designer was invaluable throughout the course of the project from the initial behavioural specification of the RF components, over the validation of the developed VHDL-AMS models, to the review of Chapter 4 describing the outcome of this work. He also helped me,

Acknowledgements

together with Alain Vachoux and Matteo Contaldo, in the supervision of the Master's thesis project of Christian Ntongo Bazangula, who implemented the initial versions of the parametrisable divider, Σ - Δ modulator, and pulse shaper component models and performed the first FSK transmitter simulations.

I would like to thank Prof. Georg Paul from the Otto-von-Guericke-Universität in Magdeburg, Germany, for the successful collaboration on the ModelLib project and his continuous support and promotion through all these years since I took his computer science introductory course in the first year of my studies. I would like to thank my Master's students Thomas Böhm and Daniel von dem Knesebeck for their contribution to the development of the ModelLib prototype reimplementations.

I would like to thank my friends Michela Peisino and Aristeidis Matsokis for carefully proof reading my Ph.D. thesis.

This research work was primarily funded by the Hasler-Stiftung under project № 2161. My first year in the doctoral program at École Polytechnique Fédérale de Lausanne (EPFL) was made possible through the grant of a bourse d'excellence by the École Doctorale Microsystèmes et microélectronique (EDMI). Parts of this research work have been funded by Projet CIMENT (Centre inter-universitaire de recherche en Microsystèmes et Nanotechnologies EPFL – UniNE), the European Union's 7th Framework Programme (FP7) projects GALAXY (GALS interface for complex digital system integration) and CREAM (Innovative technological platform for compact and reliable electronic integrated in actuators and motors) as well as the Megawatch Project of EPFL. Without this funding, it would not have been possible to carry out this research work.

I would like to thank Marie Halm for her warm welcome from the first day on and all her understanding and continuous support for the doctoral students in the administrative processes related to the EDM and beyond that. In this context, I would also like to thank Prof. Jacques Giovanola and Dr Verity Elston from the École Doctorale as well as Dr Wajd Zimmermann from the Association du Corps Intermédiaire de l'EPFL (ACIDE) for their constant strive for improving the situation of doctoral students at EPFL. These thanks go also to all the members of the Commission des Doctorants (ComDoc) de l'ACIDE. Many of them, I can now count as friends. It has been a great pleasure and fun for me spending time together with you organising so many social events for us Ph.D. students from barbecues, hikes, and ski weekends to ballroom dancing to get out of the anonymity of our labs.

I would like to express my gratitude towards all former and current members of the Laboratoire de Systèmes Microélectroniques (LSM). Amongst them are: Hossein Afshari, Omer Can Akgün, Panagiotis "Takis" Athanasopoulos, Stéphane Badel, Vahid Majidzadeh Bafar, Giulia Beanato, Alessandro Cevrero, Radisav Cojbasic, Olivier Croset, Nilay Däğtekin, Davide Garetto, Christophe Guillaume-Gentil, Frank Kagan Gürkaynak, İlhan Hatırnaz, Sylvain Hauser, Neil Joye, Nikola Katic, Gözen Köklü, Thomas Liechti, Serge Lopez, Paul Muller, Nuria Pazos-Escudero, Davide Sacchetto, Miloš Stanisavljević, Fengda Sun, Seyed Armin Tajalli, Roman Benoît Tänzer, Yüksel Temiz, Zeynep Toprak-Denis, and Michail Zervas. I am thankful to Dr Alexandre Schmid for his efficient administration of the lab operations and computer resources. I would like to thank the current and former secretaries of the lab Patricia Vonlanthen, Silvia "Sly" Hirschi Dunmore, Séverine Eggli, Valérie Marguerite Aquet for their administrative support. I enjoyed the good working atmosphere in the lab through all these years and want to thank all of you for the good times we had inside and outside of the lab.

I am particularly grateful to all my dear friends that I got to know over the past six years here in Lausanne, and which made it such an enjoyable and unforgettable experience for me. Elena Artonovska, Haykel Ben Jamaa, Pamela Collins, Vera Janowski, Neil Joye, Tobias Kober, Severin Leven, Aristeidis Matsokis, Amin Rostamian, Sébastien Rumley, Pamela Sanchez, and Miloš Stanisavljević: I cannot count the number of funny lunch breaks and other great moments on ski slopes, hikes, and other outings that we spent together and which were always accompanied by lively and inspiring discussions.

You became my closest friends here. Fedor Bezrukov, Vincent Held, Martin Hutle, Anders Sandholm, Sylvie Rockel, and Eric Lumis: we spent together so many hours climbing together indoors and outdoors. Philippe Curtet and Paul-Louis Meylan: you introduced me to the joys of scuba diving and helped me to improve my Vaudois accent. Virginie Fracheboud: out of the mutual interest in each others language grew a deep friendship, which I do not want to miss. Anja Kunze, Dominique Zosso, and your twins Florentin and Valentin: you have been close friends of mine since the beginning and I enjoyed many nice moments with you. Enrica Alasonati, Bojana Apostolovic, Yadira Arroyo, Camille Courtois, Emile Dupont, Sven Gowal, Murielle Goy, Estelle Guy, Ruud van Heeswijk, Carly Huitema, Nikola Knežević, Thomas Liechti, Sylvain Maréchal, Maria Mateescu, Melisa Nazan Kocabiyik, Brammert Ottens, Chiara Paderno, Amanda Prorok, Sébastien Quatrefages, Camille Raillon, Ali Saffarpour, Maya Shevlyakova, Jelena Stojadinovic, Nevena Vratonjic, Regina Witter, Vlasta Zavadova, and Wilhelm-Jan Zwanenburg: we have spent many nice times together. The time and activities I spent with all of you compensated so well my isolation and lack of movement in front of my computer. Axel Franke, Nadine Fröhlich, Thomas Löffler, Matthias Kriegel, Volker Peters, Felix Petri, Susann Sirotkin-Schlegel, Christian Strube, Anna Todinova, Steffen Toscher, Matthew Wasko, Philipp Weispfenning, Hannes Willeck, Claudia Wittenberger, Falk Zimmermann: thank you for staying in touch with me as good friends despite the long distances between our places of living. However, this long list of friends would not be complete without the name of Michela Peisino. I am deeply grateful to you for all your support and understanding throughout all the phases of my thesis writing. You helped me more than you can possibly imagine by always being there for me giving me love and stability throughout the difficult last year.

Last but not least, I am deeply grateful towards my family—especially, my parents Manfred and Gabi, my grand parents Anne, Werner[†], Ida, and Heinz as well as my brother Steffen—for their love, patience, encouragement, and understanding during all these years. Thank you for easing so much my life with your constant presence and care.

Lausanne, 21 March 2011



Torsten Mähne

Contents

Résumé	i
Zusammenfassung	iii
Abstract	v
Acknowledgements	vii
1. Introduction	1
2. State of the Art	7
2.1. Heterogeneous System Design Environments	7
2.2. Application of Modelling Languages to Hardware Design	8
2.3. Overview of AMS extensions to SystemC	11
2.4. Unified Description of Multiphysical Systems with Bond Graphs	16
2.5. Conclusions	17
3. ModelLib: A Web-Based Platform for Collecting Behavioural Models	19
3.1. Introduction	19
3.2. Use Cases and Requirements for a Model Library	20
3.3. Implementation of the ModelLib Prototype	25
3.4. Fine-Grained Access Control Mechanism for the Meta Information	28
3.5. Towards a 3-Tier Reimplementation of ModelLib	31
3.6. Conclusions and Outlook	34
4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems	37
4.1. Introduction	37
4.2. Modelling Methodology	39
4.3. Modelling the Frequency Synthesiser	40
4.3.1. Specification of the Frequency Synthesiser Behaviour	40
4.3.2. Specification of the Voltage Controlled Oscillator Behaviour	42
4.3.3. Design and Implementation of the Component Models	45
4.3.4. Validation of the Frequency Synthesiser Component Models	53
4.4. Application of the RF_TRX Library to the Design of a Binary FSK Transmitter	58
4.4.1. Implementation of the FSK Transmitter Model	58
4.4.2. Top-Down Design Exploration for Different Target Specifications	60
4.4.3. Bottom-Up Verification of a Design Case Implementation	65
4.5. Conclusions and Outlook	67

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling	69
5.1. Introduction	69
5.2. Overview on the OSCI SystemC AMS extensions	73
5.2.1. Timed Data Flow Model of Computation	75
5.2.2. Structural refinement using the LSF and ELN Models of Computation	80
5.2.3. Conclusions about the OSCI SystemC AMS extensions	81
5.3. Modelling Multiphysical Systems on Different Abstraction Levels	84
5.3.1. Using Domain-Specific Modelling Primitives	84
5.3.2. Using Generic Bond Graph Primitives	84
5.3.3. Using a Block Diagram	88
5.4. Integrating Dimensional Analysis with SystemC AMS extensions	91
5.4.1. Compile-time dimensional analysis with Boost.Units	92
5.4.2. Facilitating the Debugging of Errors Related to Quantity Types	95
5.4.3. Using Quantity Types in SystemC Models	99
5.5. SystemC AMS extensions eXperiments Library	100
5.6. Generic TDF Modules for Common Block Diagram Primitives	102
5.6.1. Implementation	102
5.6.2. Application Example	103
5.7. SCAX Bond Graph (BG) MoC for the SystemC AMS extensions	109
5.7.1. Requirements for the BG MoC	109
5.7.2. Architecture of the BG MoC	110
5.7.3. Module Layer of the BG MoC	112
5.7.3.1. Overview on the Classes implementing the Module Layer	113
5.7.3.2. Definition of Physical Domains	116
5.7.3.3. Implementation of Non-Conservative and Conservative BG Modules	117
5.7.4. Elaboration and Simulation Cycle of the Bond Graph Model of Computation	120
5.7.4.1. Elaboration of BG Models	120
5.7.4.2. Simulation of BG Models	127
5.7.5. Application Examples	129
5.7.5.1. Electromechanical Transducer with Linked Micromechanical Resonator	129
5.7.5.2. Interaction of the BG, TDF, and DE Models of Computation	137
5.7.5.3. Treatment of Algebraic Loops	138
5.8. Conclusions and Outlook	143
6. Conclusions and Outlook	147
A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library	151
Bibliography	161
Curriculum Vitæ	177

List of Figures

1.1.	General architecture of a heterogeneous System on a Chip (SoC).	2
1.2.	V-model of the design process of a technical system.	3
1.3.	Receiver front-end architecture of a communication link.	5
2.1.	Usage of modelling and verification languages in the SoC design process.	9
2.2.	Transformation of an electrical circuit to an equivalent causal bond graph.	17
3.1.	Use cases of a model library.	21
3.2.	Architecture of the ModelLib prototype.	26
3.3.	Entity-Relationship diagram of the meta information database.	27
3.4.	Meta information about a selected model.	29
3.5.	Meta information about design languages.	30
3.6.	Meta information about design tools.	30
3.7.	Implementation of the tuple-wise access control mechanism.	32
3.8.	Access rights inheritance between the tables.	32
3.9.	Proposed architecture for the Java EE reimplementaion of ModelLib.	33
3.10.	Refined architecture for the Java EE reimplementaion of ModelLib.	34
4.1.	Architecture of an RF transceiver for frequency-modulated signals.	38
4.2.	Global structure of the frequency synthesiser.	41
4.3.	Resonant elements of the VCO.	43
4.4.	Wiring of the varicap inside the VCO and resulting input/output impedances.	43
4.5.	Fitting of the varicap density and its derivative to the TSMC 0.18 μm process.	44
4.6.	VCO models with transient outputs.	49
4.7.	Simulation results of the test bench for the VCO model with differential digital output.	56
4.8.	Validation of the behavioural VCO model against its circuit level implementation.	59
4.9.	Structure of the binary FSK transmitter model.	60
4.10.	Organisation of the FSK transmitter test benches.	61
4.11.	Transient analysis of the binary FSK transmitter model.	62
4.12.	Frequency spectrum of the FSK transmitter output signal.	63
4.13.	Eye diagram of the FSK transmitter output frequency.	64
4.14.	Bottom-up verification of the CP and VCO circuits with the FSK transmitter model.	68
5.1.	Architecture of the OSCI SystemC AMS extensions 1.0 standard.	74
5.2.	TDF model of a simple RF front end connected to a DSP.	75
5.3.	Linear Signal Flow (LSF) model of a first-order low-pass filter.	81
5.4.	Electrical Linear Network (ELN) model of a first-order low-pass filter.	81
5.5.	Models of an electromechanical transducer linked to a micromechanical resonator.	85
5.6.	The tetrahedron of state.	87
5.7.	Interpretation of a bond as a bilateral signal flow.	90

List of Figures

5.8. Car wheel model of an electronically controlled suspension system.	91
5.9. Architecture of the <code>bufilt</code> utility.	96
5.10. Class diagram of <code>bufilt</code> 's data model to represent a unit.	97
5.11. Schematic of the electromechanical transducer block diagram model for the TDF MoC.	105
5.12. Integration of the SCAX library into the architecture of Fraunhofer's SystemC-AMS.	111
5.13. Diagram of the classes constituting the module layer of the Bond Graph (BG) MoC.	114
5.14. Overview on the modelling, elaboration, and simulation phases for the BG MoC.	122
5.15. Diagram of the classes related to the view layer and the solver layer of the BG MoC.	123
5.16. Schematics and simulation results of the electromechanical transducer example.	130
5.17. Dependency graphs of the electromechanical transducer BG models.	132
5.18. Simple model of a sensor to measure the velocity of a mechanical resonator.	139
5.19. Simple electrical example yielding a bond graph with an algebraic loop.	141
5.20. Simple electrical example yielding a bond graph with two algebraic loops.	142

List of Tables

4.1. Overview on the architectures of the VCO model instantaneous frequency output. . . .	51
4.2. Overview on the RF_TRX model library.	54
4.3. VCO model simulation performances for different architecture and output options. . .	55
4.4. Simulation performances of the FSK transmitter model variants.	65
5.1. Power and energy variables for different physical domains.	87
5.2. Basic primitives of a bond graph with their legal causality assignments.	89
5.3. Controlled junctions in hybrid bond graphs for modelling discrete switching.	90
5.4. Compilation/execution performances of the electromechanical transducer models. . . .	134
A.1. Organisation of the SCAX library implementation into several namespaces.	151
A.2. Generic waveform functors provided by the <code>scax_utility</code> library.	152
A.3. Symbols used to represent the DE, TDF, LSF, ELN, and BG MoC modelling elements. . . .	153
A.4. Block diagram modules for the TDF MoC provided by the <code>scax_tdf</code> library.	154
A.5. Block diagram modules for the BG MoC provided by the <code>scax_bond_graph</code> library. . .	156
A.6. Bond graph modules for the BG MoC provided by the <code>scax_bond_graph</code> library. . . .	157

List of Listings

4.1. Overall structure of the VHDL-AMS models developed for the RF_TRX library.	46
4.2. Architecture implementing the differential loop filter behaviour.	48
4.3. Entity declaration of the VCO with frequency output model.	50
4.4. Top-down detailed architecture of the VCO with frequency output model.	52
5.1. TDF model of the mixer.	76
5.2. TDF model of the low-pass filter using a Laplace transfer function.	77
5.3. TDF model of the Analogue-to-Digital Converter (ADC).	78
5.4. Structural model of the RF front end.	79
5.5. Event-controlled low-pass filter model with switchable gain.	80
5.6. LSF model of the lowpass filter.	82
5.7. ELN model of the lowpass filter.	83
5.8. Unformatted compiler error message for “quantity<si::current> i = R * v;”..	94
5.9. Simplified compiler error message for “quantity<si::current> i = R * v;”..	94
5.10. Time-dependent function module with two inputs from the scax_tdf library.	104
5.11. Electromechanical transducer module using the scax_tdf library.	106
5.12. Test bench for the electromechanical transducer and the mechanical resonator.	108
5.13. Physical domain traits class.	117
5.14. Trapezoidal integrator block diagram module for the BG MoC.	119
5.15. Modulated 2-port transformer bond graph module with TDF input for the BG MoC. . .	121
5.16. Generic 2-port C-field transducer module.	133
5.17. Test bench of the bond graph model of the electromechanical transducer example. . . .	136
5.18. Causality assignment results for the bond graph depicted in Figure 5.20b.	144
5.19. Dependency graph analysis results for the bond graph depicted in Figure 5.20b.	144

List of Acronyms

AC	Alternating Current
ACID	Atomicity, Consistency, Isolation, and Durability
ACIDE	Association du Corps Intermédiaire de l'EPFL
ACL	Access Control List
ADC	Analogue-to-Digital Converter
ADMS	ADVance MS™
AK	Arbeitskreis
AM	Amplitude Modulation
AMS	Analogue and Mixed-Signal
AMSWG	AMS Working Group
APF	All-Pass Filter
API	Application Programming Interface
ASK	Amplitude-Shift Keying
ASM	Abstract State Machine
BB	Base Band
BD	Block Diagram
BER	Bit Error Ratio
BG	Bond Graph
BPF	Band-Pass Filter
CAD	Computer Aided Design
CAO	Conception Assistée par Ordinateur
CGS	Centimetre Gram Second system of units
CMOS	Complementary MOS
CP	Charge Pump

CPU	Central Processing Unit
CS	Clocked Synchronous
CSEM	Centre Suisse d'Électronique et de Microtechnique SA
CSP	Communicating Sequential Processes
CT	Continuous-Time
CTRL	ConTRoL
DAC	Digital-to-Analogue Converter
DAE	Differential Algebraic Equation
DC	Direct Current
DE	Discrete Event
DF	Data Flow
DIV	DIVider
DMD	Digital Mirror Device
DPI	Direct Programming Interface
DSL	Digital Subscriber Line
DSP	Digital Signal Processing
DT	Discrete-Time
EBNF	Extended Backus-Naur Form
EDA	Electronic Design Automation
EDMI	École Doctorale Microsystèmes et microélectronique
EJB	Enterprise JavaBean
ELN	Electrical Linear Network
EPFL	École Polytechnique Fédérale de Lausanne
ER	Entity-Relationship
ESL	Electronic System Level
FAT	Forschungsvereinigung Automobiltechnik
FE	Finite Elements
FEM	Finite Element Method

FIFO	First-In, First-Out
FIR	Finite Impulse Response
FM	Frequency Modulation
FMEA	Failure Modes and Effects Analyses
FSK	Frequency-Shift Keying
FSM	Finite State Machine
GSM™	Global System for Mobile communications: originally from Groupe Spécial Mobile
GUI	Graphical User Interface
HDL	Hardware Description Language
HPF	High-Pass Filter
HTML	HyperText Markup Language
HVL	Hardware Verification Language
HW	HardWare
IC	Integrated Circuit
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEE	Institution of Electrical Engineers
IEEE	Institute of Electrical and Electronics Engineers
IET	Institution of Engineering and Technology
IF	Intermediate Frequency
IFA	Intermediate Frequency Amplifier
I²C	Inter-Integrated Circuit bus
IIR	Infinite Impulse Response
INL	Integral Non-Linearity
IP	Intellectual Property
ISM	Industrial, Scientific, Medical
ISO	International Organisation for Standardisation
Java EE	Java Platform, Enterprise Edition

JSP	JavaServer Pages
KCL	KIRCHHOFF's Current Law
KPN	Kahn Process Network
KUL	Katholieke Universiteit Leuven
KVL	KIRCHHOFF's Voltage Law
LF	Low Frequency
LNA	Low-Noise Amplifier
LO	Local Oscillator
LPF	Low-Pass Filter
LRM	Language Reference Manual
LSF	Linear Signal Flow
LSM	Laboratoire de Systèmes Microélectroniques
LTF	Laplace Transfer Function
LVS	Layout-Vs.-Schematic
MASH	Multi-stAge-noise-SHaping
MEMS	Micro-Electro-Mechanical System
MOEMS	Micro-Opto-Electro-Mechanical System
MGC	Mentor Graphics, Inc.
MIX	MIXer
MNA	Modified Nodal Analysis
MoC	Model of Computation
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
NLN	NonLinear Network
ODE	Ordinary Differential Equation
OSCI	Open SystemC Initiative
OVM	Open Verification Methodology
PA	Power Amplifier
PDA	Personal Digital Assistant

PDE	Partial Differential Equation
PFD	Phase/Frequency Detector
PLL	Phase-Locked Loop
PM	Phase Modulation
PN	Petri Net
PoC	Proof of Concept
POTS	Plain Old Telephone Service
POW	POWer management
PPF	Poly-Phase Filter
PS	Pulse-Shaped
QO	Quartz Oscillator
RAM	Random Access Memory
RDBMS	Relational Database Management System
RF	Radio Frequency
ROM	Reduced-Order Modelling
RSSI	Received Signal Strength Indication
RTL	Register Transfer Level
RX	Receiver
SAE	Engineering Society For Advancing Mobility Land Sea Air and Space International
SAW	Surface Acoustic Wave
SCA	Sneak Circuit Analysis
SCAP	Sequential Causality Assignment Procedure
SCAX	SystemC AMS extensions eXperiments library
SDF	Synchronous Data Flow
SDL	Specification and Description Language
SDR	Software-Defined Radio
SFDR	Spurious-Free Dynamic Range
SI	Système International d'unités

SINAD	Signal-plus-Noise-plus-Distortion to Noise-plus-Distortion ratio
SiP	System in a Package
SoC	System on a Chip
SOI	Silicon On Insulator
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SR	Synchronous Reactive
SRD	Short Range Device
SS	State Space
STL	Standard Template Library
SW	SoftWare
TDF	Timed Data Flow
TLM	Transaction-Level Modelling
TRX	Transceiver
TX	Transmitter
TSMC	Taiwan Semiconductor Manufacturing Company Limited
URL	Uniform Resource Locator
UVM	Universal Verification Methodology
VCD	Value Change Dump trace file format
VCO	Voltage Controlled Oscillator
VDA	Verband der Automobilindustrie
VPI	Verilog Procedural Interface
WDF	Wave Digital Filter
XML	eXtensible Markup Language

1. Introduction

Systems on Chip (SoCs) and Systems in Package (SiPs), as combinations of computer/communication hardware and software equipped with autonomy based on perception, cognition, and control capabilities, are key parts of a perpetually broadening range of applications, from chip cards and mobile phones to cars and industrial equipment. The design of such systems has currently to address a number of significant issues, namely:

Increasing complexity, due to the integration of significant computing and communication power (intelligent systems).

Significant heterogeneity, due to the variety of integrated components (analogue/RF/digital hardware, embedded software, sensors, actuators), which need to interact closely to achieve an optimal design with intertwined functionality.

Increasing environmental awareness, due to energy saving, battery operated systems, environmental monitoring capabilities, and continuous interaction with the working environment.

Increasing impact of modern silicon technologies, due to deep sub-micron and nanometer technological processes.

Increasing re-use of subsystems, due to the pressure for an ever shrinking time to market and rapid product obsolescence.

The fast progressing advances in manufacturing technology allow the integration of more and more functionality from different disciplines into a single complex heterogeneous SoC (Figure 1.1). This leads to a continuous growth in the needed design effort, where at the same time product cycles get shorter. The resulting increase in the “design productivity gap” is especially notable in semiconductor industry. There, the technological production capacity (measured by the number of available transistors) has increased since 1985 yearly between 41 % and 59 %, whereas the design capacity (measured by the efficient use of transistors) has increased only at a yearly rate of 20 % to 25 % [128]. To allow the control of the design costs and prevent them to get prohibitively expensive, new design technologies have to be continuously introduced, like block reuse and retargeting or IC implementation tools.

Analogue and Mixed-Signal (AMS) SoCs are a predominant part of today’s embedded systems used in the telecommunication, automotive, and multimedia application areas. Analogue and RF parts are playing key roles in these systems, as they are responsible for the interface to the environment (signal conditioning and communication). The steady advances of the technology will continuously broaden their application range in the future.

The design of heterogeneous AMS systems is still a highly manual work and not as automatised as the design of digital systems, which can rely on logic synthesis and place & route tools. Heterogeneous designs require a multidisciplinary approach. This imposes a diversity of description formalisms, also called *Models of Computation* (MoCs), analysis and simulation methods provided by specialised simulators for different physical disciplines and levels of abstraction, as well as CAD/EDA tools for the design and layout of the physical realisation of each heterogeneous system component. For example, the

1. Introduction

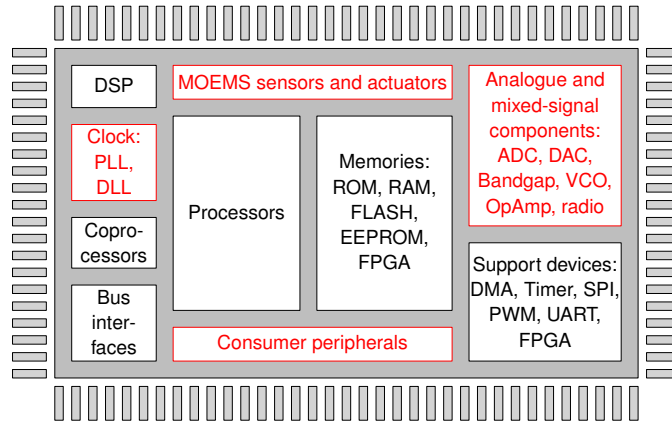


Figure 1.1.: General architecture of a heterogeneous System on a Chip (SoC). The components containing AMS or multiphysical behaviour are highlighted in red.

design of a typical Micro-Electro-Mechanical System (MEMS) like an inertial yaw rate sensor [108] requires for example:

- Optimisation and characterisation of the micromechanical resonator and (separately) of the electrostatic field distribution of the comb drive structures, which are driving and sensing the movement of the flexible structure. This is done with the help of an FEM tool like ANSYSTM from mechanical engineering.
- Simulation of the whole system on the circuit level, taking into account the coupling between mechanical and electrostatic domain within the MEMS transducer and feedback from the analogue and digital driving and sensing circuits. For this behavioural simulators and modelling languages like VHDL-AMS from electrical engineering are employed.
- Layout of the mechanical structure and of the electronic circuits. This is carried out with the help of IC layout tools.

These tools originate from different engineering fields, which leads to problems when exchanging models and other design data. The designers are forced to bridge the gaps between tools and methodologies using manual conversion of models, proprietary tool couplings and tool integrations. It is still difficult to *simultaneously* handle all the different design aspects of a mixed-signal system. An efficient tool support for the AMS design flow (Figure 1.2) is thus missing and rendering it overly complicated, error-prone, and time consuming.

In the short term, the challenge for the EDA industry is to improve the links between the existing tools. One research area, which relates to the given yaw rate sensor example, is the development of reduced-order modelling methods that allow the extraction of fully coupled behavioural models for circuit-level simulation from detailed FE models of the device [108]. In the long term, new design methods and integrated tool chains are needed to support the whole process of specification, design, integration, validation, verification, and integration of the components of a complex AMS system. First approaches for the specification, synthesis, and automated layout generation exist for moderate-complexity analogue circuits (device count less than 100), e.g., the *AMGIE* approach and the *Mondriaan* tool described in Van der Plas, Gielen, and Sansen [171].

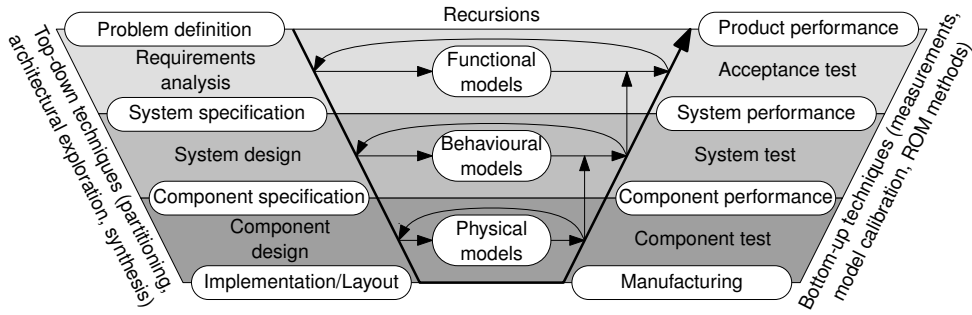


Figure 1.2.: V-model of the design process of a technical system.

Another very important issue is the capability to perform *efficient overall system verification* in the early phases of the design process. System verification is based on the development of virtual prototypes of the complete heterogeneous system and its main goals are to support use cases such as architecture exploration, performance estimations, validation of reused parts, verification of the interfaces between RF, analogue, and digital parts, verification of the interoperability with other systems, and assessment of the impacts of the future working environment and the silicon technologies used to realise the system.

Modelling tasks are therefore at the heart of the SoC design process, so it is of paramount importance that appropriate modelling languages are consistently used throughout the design process. Hardware Description Languages (HDLs) such as VHDL [77] or Verilog [78] are today routinely used for the top-down digital design process from Register Transfer Level (RTL) descriptions to the gate-level implementation and design flows based on these languages are fairly well established. At system level, the C++-based design language *SystemC* [79], promoted by the Open SystemC Initiative (OSCI) [82], has become a standard language for supporting more abstract discrete event modelling of both hardware and software components. The quest for more efficient simulation in conjunction with the increasing importance of embedded software lead in the recent years to the establishment of C-based design methodologies at system level with strong links to HDL-based design methodologies to allow a seamless path to implementation.

The situation is however more critical when it comes to the *concurrent design* of electrical or non-electrical, mixed-technology components such as analogue, RF, and MEMS blocks. Analogue and mixed-signal extensions to HDLs such as VHDL-AMS [76] or Verilog-AMS [2] have been developed to address this issue. However, they do not provide enough flexibility at system level to support in parallel different kinds of Models of Computation (MoCs) for an adapted description of the different system components including embedded software. They also do not provide enough simulation efficiency for validating complex heterogeneous SoCs. Tools like Matlab/Simulink do offer a partial alternative at system level but they do not provide a complete seamless path to the design of hardware circuits and systems. Since SystemC has established itself as an efficient solution for designing digital SoCs at the system level, it is a logical step to augment its capabilities to also support mixed-signal and mixed-technology SoCs. This is an ongoing process, which has been notably driven throughout the past years by the members of the OSCI AMSWG in form of the standardisation of SystemC AMS extensions [131]. This simulation framework is currently establishing itself for the abstract modelling and efficient simulation of complete heterogeneous systems. It is able to use in parallel different MoCs for an adapted description of the individual system parts.

Behavioural modelling of analogue/RF/MEMS blocks is a methodology that is being increasingly used in industry today as part of the design process for integrated systems. It is facilitated through

1. Introduction

the abovementioned standardised mixed-signal and mixed-technology behavioural HDLs, which are implemented by several commercial and open source simulators. Behavioural models describe the functionality of a component as input-output behaviour augmented with major non-idealities of real implementations, but without requiring a complete description of all implementation details. The purpose is mostly to verify the correct functionality of the system in acceptable CPU times by replacing (some) analogue/RF circuit schematics or 2-D/3-D models with behavioural models. The usage of behavioural models is beneficial on both sides of the V-shaped design process (Figure 1.2):

- During *top-down design*, to develop a system architecture that meets the system specifications. Although automatic formal refinement or synthesis methods for analogue/RF/MEMS systems are still mostly lacking, the existence of a *well-defined library of behavioural models* and *area/power estimation models* combined with fast high-level simulation methods may considerably help the designer during architecture exploration. Furthermore, it enables him to optimally map the top-level performance specifications onto the different blocks in the system architecture, trading off different specifications and implementation costs (e.g., area, power). An example is given in Figure 1.3, which shows a possible realisation of the receiver front-end of a digital telecommunication link and the block specification parameters, which have to be derived from the overall system specifications.
- During *bottom-up design*, to allow the overall system verification, for which the detailed circuit net lists are replaced by more abstract models that are characterised against their respective circuit implementations. The behavioural models for that kind of task may be the same as the ones used in the top-down phase or they may be different. In the latter case, they may be obtained by extracting/simplifying the circuit or device behaviour using the already mentioned reduced-order modelling methods, which involve typically symbolic equation manipulation or the generation of look-up tables.

The management of models of a device, component, or the whole system on different abstraction levels is thus an important aspect of an AMS design flow. However, the development of a model for a specific block is not an easy task because of several issues:

- The *block heterogeneity* requires specific knowledge of all the different involved physical domains.
- The model is the *formalisation of the designer's intent* and is thus requiring from him the knowledge of the modelling language, the methodology, and the used design tools.
- The model needs to be *adapted to the tools and to the specific design task*, e.g., simulation, (formal) verification, optimisation, or synthesis.
- The *abstraction level* needs to be *adapted to satisfy the requirements for a specific design task* by choosing the right trade-off between level of detail (accuracy) and execution speed of the model.

To speed up the creation of these models, it is advantageous to reuse existing models and to possibly adapt them further. Nowadays, designers often reuse only their own models or those provided by the design environment for two important reasons, which need to be addressed. First, an exchange of the models between designers is complicated by the fact that they are often not aware if and where a similar model is already existing. Second, the designer has to gain trust in the validity of the model for his specific design task, which is more difficult to achieve for foreign models because of the “Not Invented Here” syndrome. To overcome these problems, a common platform to manage, exchange, and collaboratively develop models of different formats and of different levels of abstraction is required.

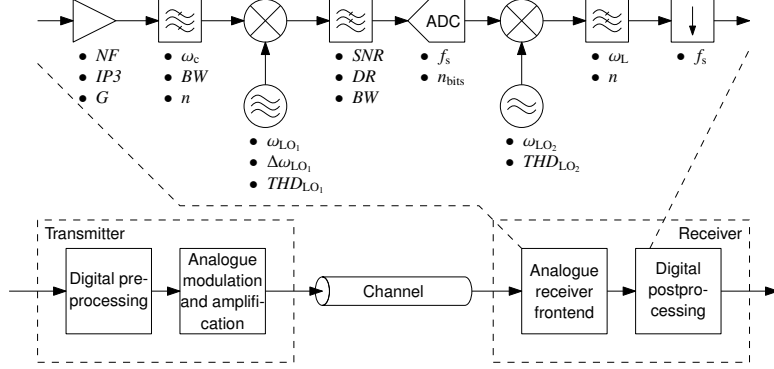


Figure 1.3.: Choosing receiver front-end architecture and deriving block specifications of a digital communication link (adapted from Gielen [63]).

The goal of the research work described in this Ph.D. thesis has been the development of an efficient modelling and simulation methodology for the design of heterogeneous mixed-signal SoCs to solve the modelling and simulation challenges mentioned above in three thematic areas that build on each other. The first problem addressed was how to improve the reuse of existing models in the complex design process for heterogeneous system, which involves so many different tools. This is a question of efficient organisation and documentation of model collections, which needs to be supported by a proper tool-independent library infrastructure. This infrastructure needs to facilitate the access to the stored models by guiding the user to find a suitable model for his current design task at hand. Additionally, it needs to provide appropriate protection mechanisms for the Intellectual Property (IP) stored in this library. The results of this work are presented in Chapter 3.

Building on this, the second problem to address was the definition of a proper modelling methodology for creating flexible models at different levels of abstraction, which can augment each other and be reused in the different phases of the design process. This has been exercised with the development of a novel library of VHDL-AMS models for supporting the top-down design and bottom-up verification of complex RF systems. The results are presented in Chapter 4 and demonstrate the capabilities and limitations of “classical” AMS-HDLs in terms of abstraction and simulation performance.

Therefore, the third and last problem to address was the need to further raise the abstraction level for the description of complex and heterogeneous AMS systems to enable their efficient simulation using different synchronised MoCs. This work is founded on the C++-based simulation framework SystemC with its AMS extensions. This popular framework for the HW/SW co-design of mainly electrical AMS systems is missing modelling capabilities to efficiently describe energy conserving multi-domain systems in a formal and consistent way at a high level of abstraction. Suitable modelling formalisms needed to be identified and efficiently supported in this simulation framework to enable the description of the non-conservative and conservative continuous-time behaviour. These have been found in form of the block diagram and bond graph formalisms. Their application to multiphysical systems modelling requires a stricter specification of the interfaces and a stricter definition of their model equations to not lose the link to the physical domain despite the generic nature of these formalisms and to enable formal assembly and equation coherency checks. The found solution based on the systematic usage of quantity types and automated dimensional analysis will be described. To efficiently support the two new modelling formalisms, a dedicated MoC has been developed. The experiences gained from the development of the RF systems modelling methodology greatly helped in the definition of flexible

1. Introduction

modelling primitives for the SystemC AMS extensions on higher levels of abstraction, which constitute the user interface to the newly proposed modelling formalisms within this simulation framework. For this task, dedicated techniques had to be developed, which enable the writing of abstract, flexible, and reusable models despite the very strict interface definition. The developed solution offers the possibility to parameterise their behaviour *and* interface in a well-defined manner. The results of this work are presented in Chapter 5. The novelty of this work lies in the extension of the SystemC AMS simulation platform with new modelling capabilities that uniformly integrate all the aspects mentioned above. It achieves the fusion of normally very opposing requirements: *genericity* to privilege reuse and *precision in the specification* to privilege verification.

The next chapter will review the state of the art in the field of modelling and simulation methodologies for heterogeneous systems, which motivated and influenced the presented research work.

2. State of the Art

Each new technology step speeds up the integration of more and more functionality into a SiP or SoC. These advancements have supported the move from digital-only to heterogeneous mixed-signal SoCs. The goal is to not only integrate the data-processing as much as possible but also to add the sensing (e.g., Analogue-to-Digital Converters (ADCs), sensors) and actuating interfaces (Digital-to-Analogue Converter (DAC), Power Amplifier (PA), switches) to the environment. Through the progress in the microsystem technology [119], these mixed-signal components are not any more limited to the electrical domain. Micromechanical, microfluidical, or microoptical components such as inertial sensors, Surface Acoustic Wave (SAW) filters, inkjet heads, pressure sensors, Digital Mirror Devices (DMDs) are just a few examples. These parts of a SoC are becoming increasingly expensive in terms of area and power consumption as they cannot be scaled-down as much as digital parts. Additionally, design times are larger due to limited reuse capabilities. Their design is still often treated with methods specialised to one physical domain (e.g., FEM, MATLAB/Simulink, and SPICE-like simulators), often neglecting domain interaction [115]. However, with the further downscaling of the devices due to technology advancements, the domain interactions are increasing and cannot be neglected anymore even for “electrical-only” designs, e.g., the thermal interaction between devices and the package due to self-heating [160]. One consequence of these issues are attempts to shift analogue/RF functionality to digital hardware and software. One example is the use of digital compensation techniques to handle analogue/RF non-idealities (e.g., limited ranges, nonlinearities, noise) [134]. This increasingly tighter interconnection between different design domains is becoming a distinguished aspect of today’s heterogeneous AMS SoC designs. Thus, as has been noted in Chapter 1, the efficient modelling and simulation of such kind of systems get more and more important.

2.1. Heterogeneous System Design Environments

One important effort to address heterogeneity in system design has been the development of the *Ptolemy II* software environment [49]. The Ptolemy approach is based on the hierarchical composition of models, each one being possibly modelled using a different Model of Computation (MoC). A MoC defines a modelling formalism, i.e., a graphical or textual language with its syntactical elements, a set of syntax rules and a set of computational rules that define the semantics of the model. Such models are also called executable models as they can be used for simulation, synthesis, or formal proof. The Ptolemy II environment supports several MoCs, among others:

Discrete Event (DE) MoC: is based on a discrete representation of time, which is suitable for modelling the behaviour of digital and sampled systems. The computational rules for such MoCs are formally defined in the form of an event-driven logic simulator.

Continuous-Time (CT) MoC: is based on a continuous representation of time, which is suitable for modelling the behaviour of analogue/RF systems. The computational rules for such MoCs are formally defined in the form of an equation solver or a circuit simulator.

2. State of the Art

Synchronous Data Flow (SDF) MoC: is an asynchronous message-passing MoC, in which First-In, First-Out (FIFO) queues model the communication between processes. Processes encapsulate sequential behaviours and possibly states. This MoC can be either un-timed (i.e., only the presence of an input token or input data sample can trigger a process) or timed (i.e., process inputs and outputs are streams of sampled data). This MoC is suitable for modelling signal processing applications.

The Ptolemy II environment has been extensively used for research purposes and inspired the development of other system design frameworks such as *Metropolis* [13] or *ForSyDe* [157]. These latter frameworks however deliberately limit their support to only one MoC, namely concurrent processes in *Metropolis* and purely synchronous/reactive models in *ForSyDe*. The main advantage over the Ptolemy approach is that they provide formal paths from abstract specifications to implementation through formal refinement, or synthesis, steps. The Ptolemy framework is better suited for system-level simulation and a commercial implementation is available from Agilent (formerly Hewlett Packard). The tool, called Agilent Ptolemy, provides a way for co-simulating SDF applications with analogue RF circuits [145].

MASCOT [19] has been developed as a modelling technique that integrates MATLAB [112] and Specification and Description Language (SDL) models in a co-simulation environment. It also supports a modelling methodology, for which abstract specifications are decomposed into a set of processes communicating through ideal, infinite-length, FIFOs. Processes are partitioned into control and data flow, thus associating the former to SDL computation and the latter to MATLAB computation. The underlying MoC is a variant of the SDF MoC called Composite Signal Flow MoC.

Wambacq et al. [178] describe the use of a data flow MoC and an appropriate representation of the signals as compositions of complex low-pass equivalent representations of these signals as a mean to efficiently model and simulate mixed-signal communication systems. The use of a data flow simulator for the analogue/RF part allows easy coupling with a digital simulator such as Synopsys System Studio.

Gielen [63] provides an excellent review of methods and tools that have been developed over the past 10 years at Katholieke Universiteit Leuven (KUL) for the design of mixed-signal/RF integrated circuits and systems. It clearly shows the needs to have a rich library of behavioural models of basic building blocks that support a large spectrum of abstraction levels and MoCs. It also discusses the *current lack of systematic methods to generate appropriate behavioural models of analogue/RF building blocks*. Automated methods have been developed for extracting/abstracting behavioural models from circuit net lists [126]. Such approaches fit well with bottom-up verification as the generated behavioural models have to more or less accurately represent real circuit behaviours. Although these models may also be used for top-down architectural exploration, other specific modelling approaches implementing functional behaviours with selected non-ideal effects such as phase noise or settling time can be used [44].

2.2. Application of Modelling Languages to Hardware Design

Parallel to the development of modelling representations (models of computation) and techniques to generate behavioural models, a lot of work has been done on the development of *modelling languages for hardware design*. In the 1980's, gate-level or transistor-level net lists were mostly used for logic/circuit simulation, timing and Layout-Vs.-Schematic (LVS) verification. In the 1990's, Hardware Description Language (HDL) such as VHDL [77] and Verilog [78] became the main design languages for digital hardware. In the same time, analogue and mixed-signal extensions to these languages (i.e., VHDL-AMS [76] and Verilog-AMS [2]) were developed. In the 2000's, C-based HDLs such as SystemC [20, 66, 79, 124], SpecC [57, 61], Handel-C [23, 118] started to become the main design languages as they enable the

2.2. Application of Modelling Languages to Hardware Design

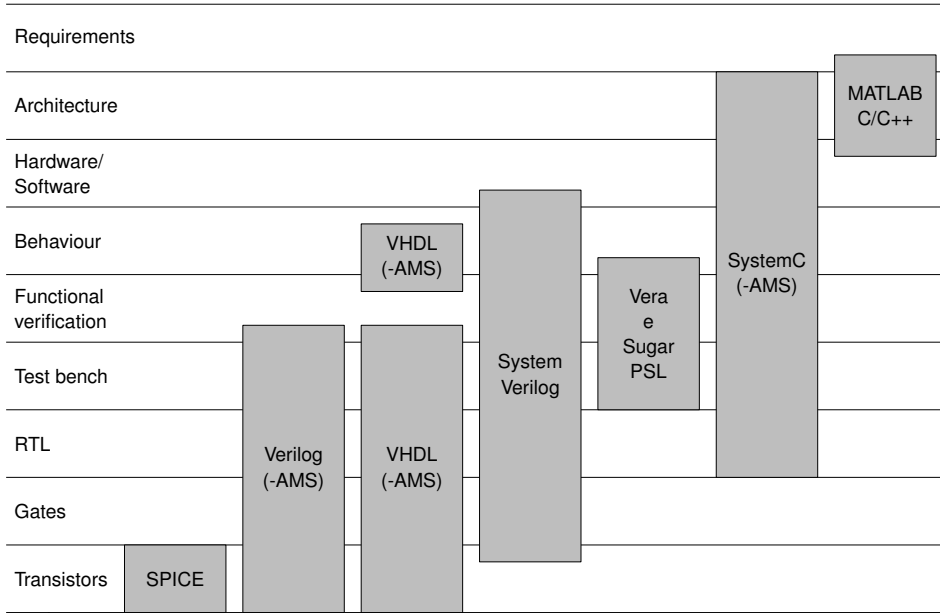


Figure 2.1.: Usage of modelling and verification languages in the SoC design process [adapted from 20].

development of more abstract models of complex systems including embedded software. Figure 2.1 shows for which tasks popular modelling and verification languages are used in the SoC design process.

SystemC [20, 66, 124] is a C++-library of classes and methods that support the description and simulation of digital Hard- and SoftWare (HW/SW) systems from functional down to register transfer level by using the *Discrete Event* (DE) *MoC*. It has seen wide industry adoption over the past decade. Its development and standardisation is coordinated by the Open SystemC Initiative (OSCI) [82]. Since 2006, it is an IEEE standard [79]. Its application domain is continuously broadening, as it is supporting powerful modelling and simulation capabilities at system level (e.g. transaction level models [62, 133]) and it is being currently extended to better support (real-time) embedded software and AMS systems. The open source and object-oriented characteristics of SystemC together with the clear separation of computation and communication in it allow for adding support of various models of computation in a layered approach [135]. For digital HW/SW systems modelling, several competing efforts provide various untimed, synchronous, and timed MoCs such as Finite State Machine (FSM), Petri Net (PN), Kahn Process Network (KPN), Synchronous Data Flow (SDF), Synchronous Reactive (SR), and Clocked Synchronous (CS). Examples of such efforts are SystemC-H [137], HetSC [71], and UMoC++ [111]. All approaches impose additional rules on how to write modules to make use of one MoC. The implementation of the MoCs themselves can greatly differ. Some are exclusively implemented as channels that use regular SystemC events to coordinate their execution with the help of SystemC’s discrete-event kernel. Some other use dedicated kernel extensions to speed up the execution of the MoCs. Patel and Shukla [136] show how these different ingredients can be combined to a successful system level design methodology for heterogeneous digital HW/SW systems.

In parallel to SystemC, SystemVerilog [80, 81] has established itself as a popular hardware description, verification, and specification language. It is an object-oriented further development of the “classic” Verilog [78] HDL to aid in the creation of complex system models on the RTL and on the architectural

2. State of the Art

level¹ and facilitate their verification using simulation and formal assertion-based methods. It includes design specification methods, an embedded assertions language, a test bench language including coverage and an assertions Application Programming Interface (API) as well as a Direct Programming Interface (DPI) to interface with IP written in foreign programming languages (using C-bindings). SystemVerilog itself focusses on digital SoC design and does not yet offer AMS extensions comparable to Verilog-AMS². The Accellera Verilog Analog Mixed-Signal Group is currently discussing how to merge SystemVerilog and Verilog-AMS [35]. In the meantime, vendor-specific co-simulation solutions are available to couple SystemVerilog with Verilog-AMS models. Compared to SystemC, SystemVerilog is a more closed environment, which hinders the integration of new MoCs into the language, which go beyond the semantics of SystemVerilog itself.

In Pêcheux, Lallement, and Vachoux [140], the modelling of an airbag system has been used to illustrate how the VHDL-AMS and the Verilog-AMS languages can be used to develop behavioural models with the necessary accuracy to validate the system's features while keeping the simulation time reasonably low (with respect to a full transistor level simulation). In addition, it showed the behavioural modelling of the chemical reaction, which inflates the airbag. This illustrated the capacity of modelling and simulating physical behaviours other than purely electrical ones and, maybe most importantly, how a multidisciplinary or multiphysical system might be modelled and simulated as a whole.

In Frevert, Haase, and Jancke [54], specific modelling and simulation methods for the design of RF systems are described. System and circuit level descriptions of various RF components are considered and combined to form system models. The presented models are written in VHDL-AMS or Verilog-A and simulated with commercial tools. A mixed-signal design flow suitable for RF systems is presented.

In Lallement et al. [100], the capability of VHDL-AMS to describe compact semiconductor models is demonstrated. A first study focuses on the EKV v2.6 MOSFET model taking into account the thermo-electrical interactions and the extrinsic aspects. The EKV model uses linearisation with respect to surface potential resulting in physically well based expressions for the whole model. A second study develops a simplified version of the MM11 Philips model, which takes into account the quantum mechanical effects. This compact MOSFET model is based on the formulation of the surface potential.

In Mähne et al. [108], the creation of virtual prototypes of complex micro-electro-mechanical transducers is presented. Creating these behavioural models can be partially automatised using a Reduced-Order Modelling (ROM) method. It uses modal decomposition to represent the movement of flexible structures. Shape functions model the energy conservation and full coupling between the different physical domains. Both modal shapes and shape functions for strain energy and lumped capacitances of the structure can be derived in a highly automated way from a detailed 3-D FE model, available from earlier design stages. Separating the generation of the reduced-order models (ROM) from the same FE model but for different operation directions circumvents current limitations of the used ROM method. These sub models are integrated into a full model of the transducer. VHDL-AMS is used to describe additional strong coupling effects between the different operation directions, which are not considered by the used ROM method itself. The application of this methodology on a commercially-available yaw rate sensor, as an example for a complex transducer, demonstrates the practical suitability of this approach.

¹Efficient system modelling on the architectural level with SystemVerilog requires the use of extension libraries like the ones developed as part of the Open Verification Methodology (OVM) [27, 28] or its successor the Universal Verification Methodology (UVM) [3, 4]. These libraries provide building blocks for well structured and reusable verification components and test environments. To this end, they also provide support for Transaction-Level Modelling (TLM).

²The Verilog-AMS language [2] is an extension of the digital Verilog language that is currently being standardised by Accellera. Verilog-AMS bears similar features with VHDL-AMS, but is mostly oriented towards circuits design.

2.3. Overview of AMS extensions to SystemC

There have been several parallel efforts to extend SystemC with analogue and mixed-signal capabilities to describe heterogeneous systems.

In Bonnerud, Hernes, and Ytterdal [22], a SystemC simulation framework has been developed that defines analog signals as new SystemC objects. The simulation semantics is still event-driven, but so called virtual clocks allow to optimise the simulation of analog and mixed-signal modules. Also, a behavioural model library of analog and mixed-signal components allows for building structural models of complex systems such as A/D converters. However, this approach has not been generalised to model any kind of analogue and mixed-signal behaviour and the use of the event-driven formalism limits its application to signal flow behaviours and fixed time step integration.

In Biagetti et al. [18], a methodology for writing models of analogue components using the SystemC standard library and simulation kernel is described. Analogue modules are implemented as regular SystemC modules with a specific architecture to handle an adaptive time step simulation. The simulation of analogue or mixed-signal models is event-driven, but each analogue block is reactivated using its own time step. Ordinary differential equations have to be manually discretised with a proper time step. This approach primarily supports signal flow modelling.

The previous approach has been extended in Orcioni, Biagetti, and Conti [129] with the goal to support the modelling of conservative systems, e.g., by including wire load effects. Their work called *SystemC-WMS* allows the implementation of analogue modules that communicate with each other by exchanging energy waves through wave channel interfaces. The wave channel interfaces are general analogue interfaces, which can support different physical domains. They allow the interconnection of modules that describe the component's physical behaviour. As only the standard communication scheme³ of the SystemC kernel is used, no modification of the SystemC library itself is necessary. The wave channel interface simplifies also the interconnection of independently developed analogue modules, because it avoids the interconnection problems commonly found in signal flow representations of conservative blocks, where the input/output role of the across (e.g., voltage) and through quantities (e.g., current) associated to the port have to be decided at implementation time of the module. This is too early because the direction of the information flow is determined from the interconnection of the modules, which is only known to the simulator at elaboration time. The wave channel methodology avoids this problem since incident waves always have the input role and reflected waves the output role. Parallel and series connection of modules can be accounted through appropriate channels, which dispatch the waves to the modules they connect together. This is similar to the scattering junction of Wave Digital Filters (WDFs). The response of a module to the incident waves is described through a, b parameters, which are part of the WDF theory. The methodology can be extended to circuits with mildly nonlinear elements. A half-bridge inverter was modelled as an application example using SystemC-WMS and simulated using a fourth-order Adams-Bashforth Ordinary Differential Equation (ODE) solver. The simulation results showed good correspondence to the ones obtained from an equivalent model created using MATLAB's Simulink Power toolbox and simulated using the ode15s stiff ODE solver. The simulation took about five times longer in SystemC-WMS. This can be partly attributed to the different ODE solvers used. Nevertheless the simulation performance of SystemC-WMS is limited to a good deal by the fact, that for each integration time step several discrete events are scheduled, which invoke the SystemC simulation kernel so that discrete and continuous parts cannot run independently from each other.

³Modules communicate in SystemC through the interface of a connected channel by invoking one of its member functions. Events inside the channel can in turn activate other modules.

2. State of the Art

In Al-Junaid and Kazmierski [92], SystemC is extended to *SystemC-A* that supports analogue variables and analogue components (e.g., SPICE-like primitives or user-defined components defining arbitrary Differential Algebraic Equations (DAEs)). Each analogue component in a netlist contributes to the setup of a Modified Nodal Analysis (MNA) system matrix by specifying the contributions of each conservative terminal to the Jacobian and to the right hand side of the DAE system. The interconnection of analogue and digital models is handled through specific interface models. Digital to analogue interaction is realised by converting a digital signal into an analogue signal using Backward Euler integration with very small time step. Analogue to digital interaction is realised by detecting the crossing of thresholds. The SystemC simulation kernel is modified to include the execution of an analogue solver. Mixed-signal timing synchronisation is achieved using a lock-step mechanism to avoid backtracking. In Al-Junaid, Kazmierski, and Wang [91] *SystemC-A* is used to model an automotive seating vibration isolation system. The case study showed good correspondence between the simulation results of two equivalent models of the seating vibration isolation system, one written in SystemC-A and the other in VHDL-AMS. However, the presented solution has the drawback that it required modifications to the standard SystemC kernel itself to couple the analogue solver with the discrete-event solver instead of providing an abstraction layer on top of SystemC to allow the parallel integration of various continuous time MoCs. Also, the way of defining contributions to the system matrix is very close to circuit level and thus may not be an appropriate approach for complex heterogeneous systems.

Vachoux, Grimm, and Einwich [169] define the context of the development of extensions to the SystemC modelling framework to support the description and the simulation of analogue and mixed-signal systems, called SystemC-AMS. In Vachoux, Grimm, and Einwich [170], the developed SystemC-AMS prototype is presented in detail. It already proved to be useful in industrial projects for the development of the specification and “golden” reference model for the implementation of a voice codec chip used in Digital Subscriber Line (DSL) modems between the digital world and the analogue Plain Old Telephone Service (POTS) [46]. Two formalisms, or MoCs, are implemented: a *timed variant of the Synchronous Data Flow (SDF) MoC* is used to model signal processing dominated behaviours as well as more general continuous-time behaviours using oversampled models and a *linear network MoC* provides a library of linear electrical primitives for describing linear macro models. Both MoCs are synchronised with the discrete event SystemC simulation kernel through a synchronisation layer, thereby allowing mixed-signal and mixed-MoC simulation.

This early SystemC-AMS prototype has been evaluated towards its applicability to the modelling of MEMS, which involve several physical domains under the rule of energy conservation laws. The prototype was examined on a heterogeneous inertial navigation system with micromechanical sensors. Two modelling approaches were reported. In the first one [109], the mechanical quantities were mapped on electrical ones, so that the linear electrical network models like resistor, capacitor, and inductor provided by SystemC-AMS could be used to represent the mechanical dampers, springs, and masses. In the second approach [110], the mechanical system was modelled using a non-conservative block diagram with feedback, which was simulated using the SDF MoC and had simulation performance advantages over the first approach. The developed models showed good conformance to the results obtained with VHDL-AMS reference models of the sensor. However, the development effort for the SystemC-AMS models was higher due to the lack of dedicated modelling capabilities for multiphysical systems. The work also showed a need to integrate such capabilities into SystemC-AMS, because the considered system contained, besides the MEMS sensor and its analogue front-end, a tightly coupled digital part with embedded software for the signal processing of the sensor signal and tuning of the sensor. For modelling this digital HW/SW part, SystemC-AMS is better suited than VHDL-AMS yielding in a considerable simulation performance advantage.

Similar findings are reported in Caluwaerts, Galayko, and Basset [30], where the modelling of an electromechanical energy harvester consisting of a resonator, a variable capacitor, a charge pump and a flyback circuit is presented. A combination of linear electrical primitives and user-defined SDF modules is used to describe the system. The nonlinear behaviour of the involved diodes is modelled as a resistor, which resistance value is controlled through an SDF module based on the sensed voltage across the resistor. This very close feedback imposes very small time steps for the transient simulation due to the required unit delay in the SDF feedback loop to keep the resulting error small. This clearly shows SystemC-AMS's current limitations regarding the modelling of physical systems due to the missing nonlinear solver and dynamic time step capabilities.

In Herrera, Villar, and Grimm [72], it is shown how to couple SystemC-AMS [170] with HetSC [71] to support in parallel a wide range of MoCs. This enables the use of SystemC for the complete specification of increasingly heterogeneous embedded systems, which include software control parts, digital hardware accelerators, analogue front-ends, etc. Semantical and syntactical issues for the cooperation of both libraries on top of SystemC are discussed with a focus on the interfaces provided between the different MoCs for their interaction during simulation. One possibility for the synchronisation of the MoCs from the two libraries is to use the DE MoC of SystemC as an intermediate layer, as both libraries already offer this synchronisation capability. The second possibility allows a direct coupling of specific HetSC MoCs and SystemC-AMS MoCs using the concept of border channels in HetSC. The paper recognises the need to standardise the synchronisation semantics between different MoCs.

A similar problem is addressed in Damm et al. [39], where it is shown how to couple SystemC-AMS models with loosely-timed TLM 2.0 models using a temporal decoupling approach with the focus on the SystemC-AMS side acting as a streaming data producer and/or consumer. Converter elements between the SDF MoC and loosely timed TLM MoC have been implemented that exploit the loosely timed coding style of TLM 2.0 to fit with SystemC-AMS's timed SDF MoC in such a way that the high simulation performance of both MoCs is preserved.

In Zaidi, Grimm, and Haase [181], it is shown how to couple the timed SDF MoC of SystemC-AMS via the Verilog Procedural Interface (VPI) with an AMS simulator to do mixed-level co-simulation of AMS systems, in which most parts of the system are modelled on the system level using SystemC(-AMS) and selected blocks are replaced through more detailed behavioural (Verilog-AMS or VHDL-AMS) models or circuit level (SPICE) models.

All three efforts [39, 72, 181] show the advantages of the SystemC-AMS architecture, which facilitates the integration of very different modelling formalisms and tools to an efficient simulation platform supporting the specification, design, and verification of complex heterogeneous AMS SoCs.

In Zhu, Sander, and Jantsch [182], a formal heterogeneous model of computation framework for SystemC is introduced that is called HetMoC. It complements the already presented approaches [71, 135, 170]. It is based on a very formal definition of the semantics for Continuous-Time (CT), Discrete Event (DE), Synchronous Reactive (SR), untimed Data Flow (DF), and SDF model domains. A new modelling style for the CT MoC is presented, which has pure CT dynamics. Based on it, the other MoCs are derived by stepwise abstraction. In this framework, the target system is modelled as a process network. Blocks are processes that specify computation and edges are signals to connect processes. For each model domain the signals, domain interfaces, and processes are defined. The domain interfaces are polymorphic allowing to combine models using different MoCs. The implementation of the HetMoC framework in SystemC is inspired by a system level functional modelling style proposed for untimed dataflow models in Grötter et al. [66]. All models are communicating through standard SystemC `sc_core::sc_fifo<T>` channels. As an application example, an adaptive Amplitude-Shift Keying (ASK) transceiver system is modelled with HetMoC and its simulation results/performance is

2. State of the Art

compared to a SystemC-AMS reference model. The results are promising even though SystemC-AMS showed a clear performance advantage paid with a higher memory consumption due to its more complex but optimised implementation. HetMoC's implementation seems to rely for the moment purely on SystemC's DE kernel to control its model execution without doing any kernel extension to optimise its model execution. This could be an explanation for the performance disadvantages over SystemC-AMS. In the perspective of the standardisation of AMS extensions to SystemC, the presented framework is interesting due to its sound formal base to integrate different modelling domains. These concepts could contribute to the definition of a clean synchronisation layer between the different CT MoCs of the AMS extensions and the DE MoC of SystemC. Also the integration of some of the described MoCs could be interesting.

The development of the abovementioned SystemC-AMS prototype [170] was accompanied by the SystemC-AMS study group [164] with the goal of generalising and standardising the concepts introduced with SystemC-AMS. With support from the semiconductor industry (notably NXP and STMicroelectronics), the study group promoted the creation of an official working group within the OSCI consortium [82] that coordinates the development/standardisation of SystemC and related libraries. Since its foundation in 2006, the charter of this OSCI AMS Working Group (AMSWG) [132] has been the development and standardisation of AMS extensions to SystemC to foster their industry acceptance. Based on the collected requirements and use cases [48], the AMSWG developed a Language Reference Manual (LRM) [130], which became in March 2010 an official OSCI standard [131]. In parallel to the standard release, the AMSWG published a user's guide [14]. Fraunhofer IIS/EAS released also a conforming Proof of Concept (PoC) implementation of the SystemC AMS extensions 1.0 [53] as a further development of its former SystemC-AMS prototype. In their first version, the AMS extensions primarily address the needs for describing the continuous-time behaviour of purely electrical AMS SoCs by proposing three MoCs, which allow their description on different levels of abstraction using *Timed Data Flow* (TDF), *Linear Signal Flow* (LSF), and *Electrical Linear Network* (ELN). This makes them well-suited for the design of communication systems, which analogue front ends are tightly coupled to complex digital control [46], and for the design of Digital Signal Processing (DSP) applications. However, their modelling capabilities are not yet well suited to describe *energy-conserving multiphysical system* components with *nonlinear behaviour* in a *formal and consistent way at a high level of abstraction*. Its requirements specification [48] already mentions these needs to enable their usage, e.g., in the automotive sector. Addressing this issue is one of the main topics of this thesis work. Therefore, the OSCI SystemC AMS extensions will be introduced in more detail in Section 5.2.

First experiments were already done with the old SystemC-AMS prototype to extend it for the modelling of conservative elements with nonlinear dynamic algebraic equations. The results have been reported in Einwich et al. [47] using a micro relay as a multiphysical example. The newly proposed nonlinear MoC defines a new module class, which provides callbacks that can be overloaded to describe the module's energy-conserving behaviour as contributions to the DAEs system of the nonlinear network formed by the interconnected modules of this type. To this end, each module defines its contribution to the through values (currents) of the nodes connected via the module's ports in dependency of the across values (voltages) and the derivation of the across values. Additional equations have to be described in the form $0 = F(t, v_{\text{ports}}, vars_{\text{ports}}, vars_{\text{ports}}, \dot{v}_{\text{ports}})$. The proposed syntax resembles Verilog-AMS, but in a way that it conforms to the syntax of C++. The advantage of the proposed approach is in its modularity. The nonlinear MoC integrates itself into the infrastructure of SystemC-AMS without requiring modifications to the latter. It uses the synchronisation layer of SystemC-AMS to seamlessly interface with the SDF MoC of SystemC-AMS and the DE MoC of SystemC. Thus, large heterogeneous systems can be simulated, which components have been modelled using different MoCs in parallel.

In Uhle and Einwich [168], it is reported how the previous approach has been refined and generalised. *Natures* can now be declared like in VHDL-AMS to associate nodes, terminals, and branches to physical domains. Thus, model assembly mistakes can be detected upon compile time. The implemented syntax for describing the energy conserving behavioural has been improved to be more VHDL-AMS-like and thus user-friendly. The proposed new language constructs have been aligned with the syntax of the standardised OSCI SystemC AMS extensions. The implementation is founded on the SystemC-AMS PoC implementation of this standard. The nonlinear solver can generate events based on threshold crossing and is able to backtrack to react on events. Thus, if an NonLinear Network (NLN) model is solely coupled to a DE model, the synchronisation semantics are equivalent with the VHDL-AMS simulation cycle. If TDF models are additionally coupled to the NLN model, the synchronisation becomes more complex, because synchronisation can only happen at the end of a TDF cluster period. The capabilities of this NLN MoC are demonstrated on a complex electromechanical window lifter model coupling the electrical, mechanical, and magnetical domains. For a pure NLN model, both approaches [47, 168] cannot achieve considerable runtime advantages over an equivalent VHDL-AMS/Verilog-AMS model, as the underlying DAE system and nonlinear solver algorithms are similar. However, in the nowadays typical cases where the complexity of the digital HW/SW part dominates over the analogue part, the system simulation clearly profits from the more abstract modelling capabilities offered by SystemC and its AMS extensions.

In Hartmann et al. [70], the modelling of physical control systems with SystemC-AMS is described. A model of a crane with embedded control is used, which has been already previously proposed as a system modelling benchmark [122]. Due to numerical stability issues, an external 4th-order Runge-Kutta solver is integrated into the simulation replacing the linear State Space (SS) solver provided by SystemC-AMS. The approach is interesting, because the external solver is encapsulated in a way that it provides the same interface as the original SS solver and thus requires only minimal modifications to the model itself.

Like in the “classical” AMS-HDLs (e.g., VHDL-AMS and Verilog-AMS) SystemC-AMS and SystemC-A use a generalisation of the classical electrical network model to represent conservative systems, in which modified Kirchoff’s voltage and current laws account for the energy conservation. The generalised networks accurately represent the physical structure of the conservative system, but they cannot visualise the computational structure⁴ of the system: Which quantities act as inputs, and which as outputs? The relation between the across and through quantities at the terminals of each component are described using ODEs or DAEs. One resulting problem is that the solvability of the implemented model can usually only be insufficiently checked at model elaboration time: e.g., VHDL-AMS only checks that the number of defined across and through equations matches, which is a necessary but not sufficient condition. In Haase [67], the problem of the possible definition of syntactically correct simulation problems without a solution is discussed from a mathematical point of view to give reasons and establish rules that may help to avoid these difficulties.

An extensive bibliography of publications related to the initial SystemC-AMS prototype and its standardised successor in form of the OSCI SystemC AMS extensions can be found on the homepage of the SystemC-AMS study group [164]. It documents its growing popularity in the research and industrial communities, who are applying both to a variety of application domains, e.g., RF systems [5, 174, 180], control systems [70], test development [102], automotive components [10, 99, 150, 168], biological labs on chip [141]. The SystemC AMS extensions have already proofed their capability as an integration platform able to support very different modelling formalisms, which can interact with each other. The

⁴The order of model equation execution can be, e.g., easily derived from the signal flow diagram in the case of a non-conservative system.

2. State of the Art

openness of the C++-based SystemC simulation framework allows an extensibility with 3rd party libraries and tools, which cannot be matched by “classical” HDLs, as they do not give the user such a deep access to the internals of the simulation mechanism. The standardisation of SystemC and its AMS extensions is ongoing. For the AMS extensions, goals are to make the TDF MoC even more flexible by supporting features such as the dynamic modification of the TDF time step or to trigger the TDF cluster execution as a reaction to events. Long term goals are to define a standard API for plugging in external solvers and the formalisation of the synchronisation layer between the CT and DE MoCs to offer a standardised API for the integration of new MoCs.

2.4. Unified Description of Multiphysical Systems with Bond Graphs

One way to unify the description of multiphysical systems is the bond graph methodology [94], which is more and more used in mechanical engineering, mechatronics, control theory, and to some extent in power electronics [9, 17]. A good overview on the bond graph notation, history, and future trends is given in Breedveld [24] together with an extensive bibliography. Bond graphs have been originally introduced by Henry M. Paynter in 1959 with the introduction of the junctions [138, 139]. The bond graph notation was refined by his students Dean C. Karnopp and Ronald C. Rosenberg [95, 96]. Rosenberg also developed the first computer tool, called ENPORT, for bond graphs [154, 155]. Jan J. van Dijk [172] and Jean U. Thoma [165] introduced bond graphs in the 1970’s in Europe. These pioneers spread the bond graph formalism world wide and applied it to different engineering domains [93]. Further resources about bond graphs, related research, and supporting tools can be found, e.g., in Cellier [31], Filippo et al. [52], and HighTech Consultants [74].

Using the bond graph formalism, each conservative system can be transformed from its domain-specific representation (e.g., electrical circuit, mechanical multi-body system, rigid bodies, fluidic networks, thermal networks) to an equivalent bond graph representing graphically the energy flow between primitives modelling energy sources (S_e , S_f), resistive/capacitive/inertial behaviour (R , C , I), quantity transformations (TF , GY), and energy distribution through junctions (0, 1). All elements can have nonlinear characteristic equations. One big advantage of bond graphs is that they can be annotated in a systematic way with the *causality* for each bond (current-out or effort-out causality), which visualises the computational structure of the bond graph and allows to sort the element equations in the right order for a fast model execution. As an example, the transformation of a simple electrical circuit to an equivalent bond graph, annotated with causality and then derived computational structure, is shown in Figure 2.2. Additionally, the assigned causalities allow some further formal checks on the model: the number of states and non-states in the system, the presence of algebraic loops during model execution, or if it is an ill-posed model. The causality assignment also allows for a good integration of bond graphs with signal flow graphs and their transformation in the latter. There are extensions to classical bond graphs, which allow, e.g., hierarchical models with more abstract *word bond graphs* [24, 94] or the handling of discrete switching due to external signals in so called *hybrid bond graphs* [16, 123]. Very similar is the concept of switching elements described in Edström, Stromberg, and Top [45]. In both cases, the challenge of local switching is the local causality change at the switched junction, which needs to be propagated throughout the whole graph and also imposes a reinitialisation of the internal states of the bond graph primitives. It is therefore still an on-going research topic how to implement these switching concepts in an efficient way that limits the negative impact on simulation performance.

Besides specialised bond graph tools such as ENPORT [155], TUTSIM [86], MTT [59], CAMP-G [29, 64], HyBrSim [123], SYMBOLS SonataTM [75], 20-sim [38] (reviews can be found, e.g., on HighTech

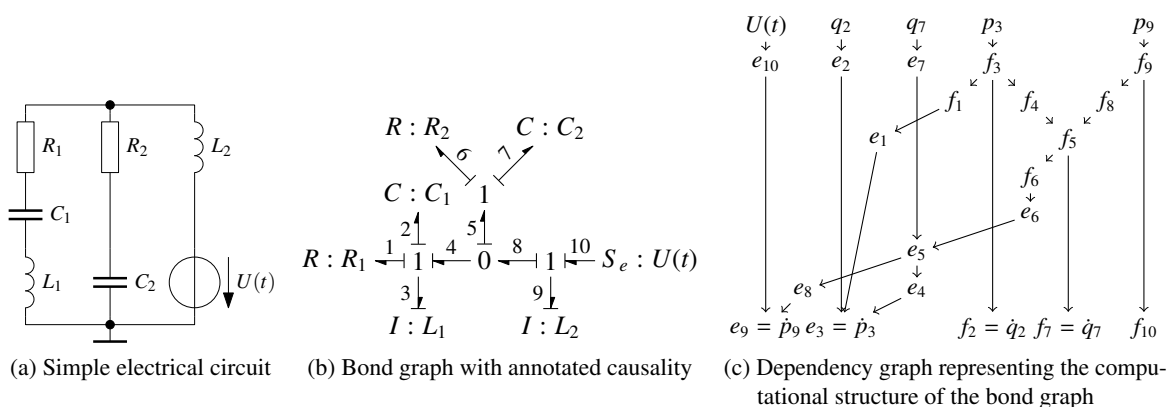


Figure 2.2.: Transformation of an electrical circuit to an equivalent bond graph annotated with causality and then derived computational structure.

Consultants [74]), there are also efforts to integrate bond graph support in widely used mathematical tools such as Matlab/Simulink (e.g., BG V2.1) [60, 112, 113] or Mathematica (e.g., Bond graph tool box for Mathematica) [176, 179]. Another approach is to use the capabilities of an AMS-HDL to represent bond graphs. In Cellier and McBride [33], it is described, how causal and acausal bond graphs are implemented in Modelica/Dymola [11, 40] for the publicly available BondLib [32]. The bond graphs can be drawn using the Graphical User Interface (GUI) of Dymola or input in a textual form. As application examples the modelling of a hydraulic motor control system and the thermal budget of the Biosphere 2 are presented [34]. However, Modelica’s capabilities are weak on the discrete event side, which is one reason why it is not used in the microelectronics community and thus not suitable for mixed-signal SoC design. In Pêcheux et al. [142], it is shown that bond graphs can be represented in VHDL-AMS and are used to model a Pb/Fe battery and a complex airbag SoC including MEMS accelerometer, digital control, thermal network, laser diode, and the chemical reaction to inflate the cushion. The authors note that the implementation of bond graphs is not very efficient because the support of VHDL-AMS for continuous systems is limited to non-conservative signal flow description based on free quantities and conservative generalised networks. Thus, from a bond graph point of view, *effort* and *flow* variables of a port have to be referred from the *across* and *through* quantities defined between two terminals. The guarantee of Kirchhoff’s voltage and current laws inside the generalised networks leads to the generation of unnecessary duplicate equations in the case of bond graphs, where these laws are already taken into account through the 0- and 1-junctions. Causality cannot be assigned to the bonds and thus not be profited from. All three aspects have a negative impact on the simulation performance of the model. The main focus of the paper is thus the coherence of VHDL-AMS with regards to bond graphs: to present a way how to shift from a common system level description to a circuit level description suitable for the mixed-signal simulators of common EDA frameworks.

2.5. Conclusions

The described properties of bond graphs make them attractive for the design and verification of heterogeneous AMS SoCs on the system level, since they unify and thus ease the description of multiphysical system components and their interactions. The possibility of doing formal checks on the system model

2. *State of the Art*

using the causality analysis is interesting, as it allows to gain insight into the kind of physical interactions of the system components and into the computational structure of the model. Their close relationship with signal flow models promises interesting simulation performance advantages over the generalised network formalism used by the classical AMS-HDLs, discussed in this chapter. As the review of the tools/languages supporting bond graphs showed, the formalism is not yet well supported by any tool/language popular for SoC design. Over the past years, the SystemC-AMS prototype and the from it evolved and standardised successor OSCI SystemC AMS extensions have shown that they are able to host different discrete-event/discrete-time/continuous-time modelling formalisms in parallel. Due to the openness of these simulation frameworks, they are gaining popularity as the central specification/integration tool for heterogeneous SoC design. They have proved themselves to be very suitable for signal processing dominated system designs, but lack frequently demanded modelling capabilities for nonlinear multiphysical systems design. This triggered the initial motivation to integrate the bond graph formalism into the OSCI SystemC AMS extensions as a new MoC, which will be presented as part of Chapter 5.

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems

This chapter describes ModelLib, a web-based platform for collecting models from different domains (e.g., electrical, mechanical, or electromechanical) and levels of design abstraction (e.g., system, circuit, or device level). Use cases for this tool are presented, which show how it can support the design process of complex heterogeneous AMS systems through better reuse of existing models for tasks like architecture exploration, system validation, and creation of more and more elaborated models of the system. The implemented ModelLib prototype is described as well as how the meta information is protected through a hierarchical access control mechanism. The discussion is concluded by presenting the first steps towards a more modular prototype implementation.

3.1. Introduction

Chapters 1 and 2 described how under the increasing market pressure AMS design flows need to be continuously optimised with the goal of reusing as much as possible internal or external designs for the new projects. They underlined the importance of using various kind of models throughout the whole design process and how challenging it is to write good models and *reuse* existing ones. Often, designers reuse only their own models or those provided by the design environment for two important reasons that need to be addressed. First, an exchange of the models between designers is complicated by the fact that they are often not aware if and where a similar model is already existing. Second, the designer has to gain trust in the validity of the model for his specific design task, which is more difficult to achieve for foreign models because of the “Not Invented Here” syndrome. To overcome these problems, the models need to be documented regarding their interface, implementation, extend of covered effects, how they were verified, and other general properties. The designer needs to understand from this information the model structure and its functionality as well as to judge if it is suitable for the design task at hand. It has to be clear, which tools the model is expected to be compatible with. The Engineering Society For Advancing Mobility Land Sea Air and Space International (SAE) covers the documentation issues in the *SAE J2546 Model Specification Process Standard* [156]. There are ongoing activities to develop collections of verified models for different design languages like *Modelica* [11] and *VHDL-AMS* [73, 117]. Those libraries are often available as archives from the Internet. They usually provide some documentation besides the model source code, which can be, to some degree, automatically extracted from the model sources using a documentation generator [51, 173].

There are also tool-specific library managers, e.g., the *Library Manager* in the Cadence™ IC Design Environment or the Workspace in Mentor Graphics™ ModelSim, to handle the various models of a project and the design kits. However, these tools cannot cope with the management of the models over the tool boundaries, as it is required for heterogeneous AMS system designs. Static websites documenting and linking to model archives are a solution, but can only partly address these issues and require a lot of manual maintenance to stay up-to-date. In Rogin et al. [153], it is described how these documentation

3. *ModelLib*: A Web-Based Platform for Collecting Behavioural Models

maintenance issues are addressed for the automotive VHDL-AMS model library under development by the VDA FAT-AK30 based on specific guidelines [68]. The model sources are centrally managed in a Subversion repository and the documentation in HTML format is extracted automatically from specially formatted comments contained in the model sources and is nightly updated. This effort is a good step into the right direction.

The *ModelLib* project described in this chapter tries to further address the described reuse issues through the development of a web-based platform with the following objectives:

- Creating a platform for collecting and distributing models from different domains (e.g., electrical, mechanical, or electromechanical) and levels of design abstraction (e.g., system, circuit, or device level) independently of the design tools;
- Improving the access to and reuse of model collections;
- Establishing a validation process for the collected models through collaborative review and development;
- Organising the meta information about a model and allowing queries on it;
- Supporting the designer's decision for the right model for each task (e.g., architecture exploration, performance analysis, verification); and
- Realising this in an open source framework, but with appropriate Intellectual Property (IP) protection for the stored documentation and models.

Section 3.2 presents the basic use cases for a model library and how it can support the work of the AMS designer. From these, requirements for the *ModelLib* platform are developed. The architecture of a prototype implementation is described in Section 3.3. Section 3.4 presents the implementation of the fine-grained tuple-wise access control mechanism to protect the IP constituted by the meta information about the models stored in the library. Section 3.5 outlines the first steps towards a more modular and feature-complete reimplementing of the *ModelLib* prototype using Java Platform, Enterprise Edition (Java EE). Conclusions are given together with an outlook on possible further research directions in Section 3.6.

3.2. Use Cases and Requirements for a Model Library

A model library like *ModelLib* can be set up on different organisational levels, like within a project group, a company, or as a community portal on the Internet. The basic use cases (Figure 3.1) for the AMS designer accessing the *ModelLib* server through a client on his computer for submitting, retrieving, and collaboratively developing the models over the Internet remain the same, while the demands for security and required detail of access control will rise with each level of broader access. The communication between client and server needs to be done through an encrypted channel. Users have to authenticate themselves in front of the model library so that the information stored in the library can be selectively made available to the different users. This is needed to protect the IP of the authors and to support the conformance to their license terms, under which they are making their work available to the public or a restricted group of users.

The users of the model library can be categorised into five different roles. Each has a different profile of allowed actions on the library (Figure 3.1). The *guest* is anonymous and has thus the minimum rights.

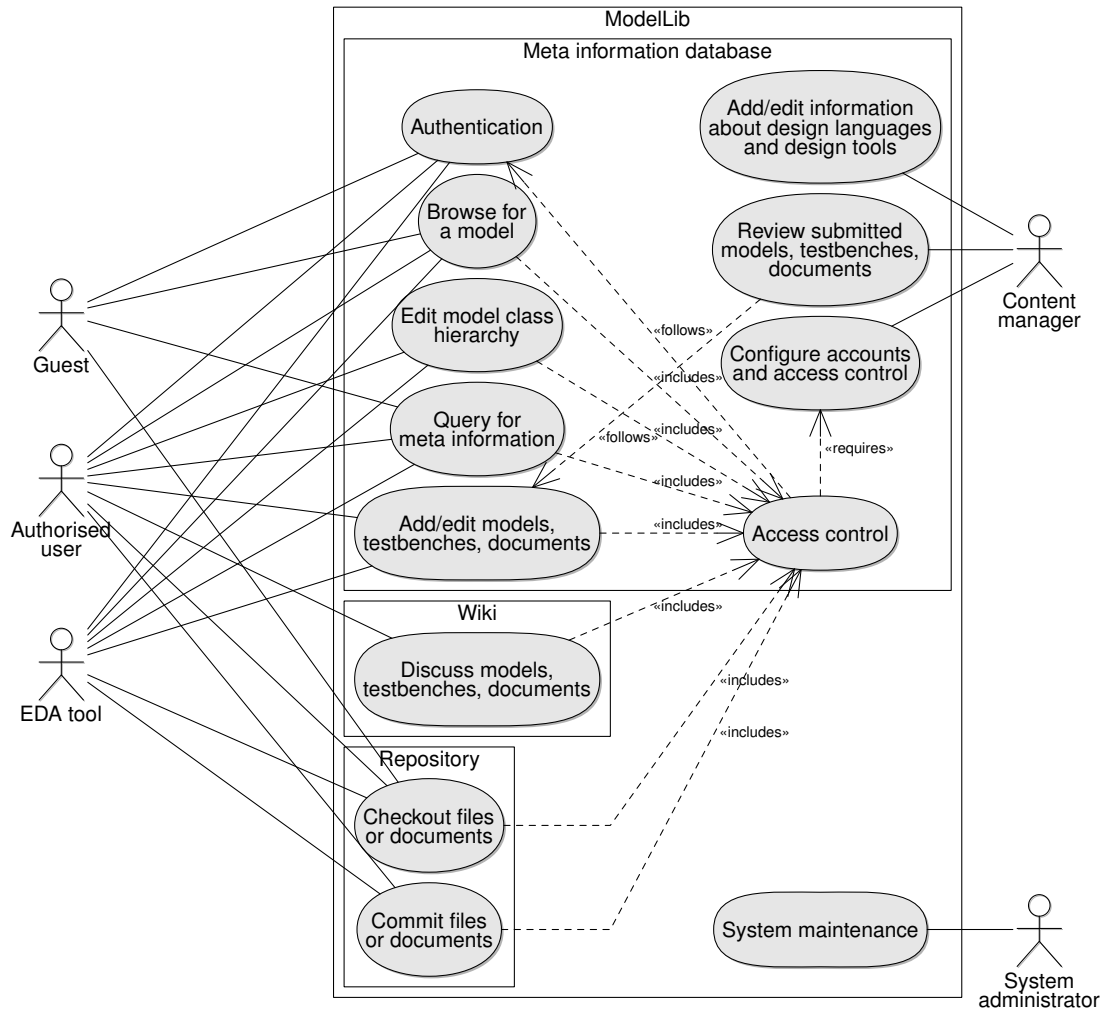


Figure 3.1.: Use cases of a model library.

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

He can only browse for a limited collection of public models, send queries about their meta information, and retrieve their source code by checking it out from a public repository. By authenticating himself with a valid user name and password, he can become an *authorised user*, who gains further rights dependent on his membership in different *groups*. These groups *grant* or *deny* him access to the different parts of the library (the different models, test benches, and documents themselves, as well as the supplementary meta information about them). He can participate in the review process by discussing the models, test benches, and documents stored in the library. He can also actively contribute to their development by submitting new models, adding/editing of the meta information about them, and organising them into different categories called *model classes*. *EDA tools* need also a direct access to the model library and are a special form of an authorised user. There are two privileged roles. The *content manager* configures the accounts and access rights of the users, reviews their submitted models, test benches, and documents, and adds commonly used information about supported design languages and tools. The *system administrator* is responsible for the maintenance and the further development of the model library platform.

In the following, the use cases for retrieving a model (through browsing or a more complex query) are discussed in more detail, as they show which meta information needs to be stored in the library alongside the models to support the designer's decisions. Then follows the description of the remaining use cases regarding the submission, the update, and the joint development of models.

One way to access the models in the library is to directly browse through the collection. For this, the available models need to be sorted into a hierarchy of *model classes*. A model can be at the same time member of different classes, e.g., to reflect that a model is part of some IP library and that it is modelling effects from a particular physical domain. In SAE [156, Appendix A] a list of automotive EE commodities structured into a hierarchy of EE Commodities, Class, Sub_Class I, and Sub_Class II is given. It illustrates the diversity of automotive EE commodities that may be subject to modelling and simulation. It could serve as a guide to structure the model library. After selecting a model, the user is presented the meta information describing its properties, which can be detailed into *structural meta information* describing “How the model is built?” and *semantical meta information* describing “How the model can be used?”. The *structural meta information* describes the following aspects:

- *Name and storage place* of the model within the model class hierarchy;
- *Interface* including detailed information about all parameters and ports as well as the assertions associated to it;
- One or more *model implementations (architectures)* in the form of behavioural description(s) and/or structural description(s) along with the assertions associated to it/them;
- *Design entities*, each one gathering the interface and one model implementation using a particular design language (e.g., VHDL-AMS) and tested against particular tools (e.g., simulators or synthesisers); and
- *Test benches* to validate some design entities. They are implemented using a particular design language version and stored in a number of files, which are known to work together with some design tool versions. Test data and expected results can be included.

To each of these aspects, an arbitrary number of references to external documents can be given. These are information, which can be directly extracted from the model sources. To support the porting of design entities and test benches to other design tools, it is required to store additional information about the different versions of available design languages and design tools, as well as which tool versions

support which language versions. Figures 3.4, 3.5, and 3.6 from the ModelLib prototype described in Section 3.3 show one way of presenting the structural meta information to the user.

The *semantical meta information* further characterises a model regarding its fidelity and performance. The *SAE J2546 Model Specification Process Standard* [156] discusses some of these aspects. It recognises that models can be classified according to their sophistication, capability, and captured intelligence. Additionally, the many dimensions of fidelity make the classification challenging, since they may be sequential, parallel, independent, contradictory, and/or redundant. Properties of a model may be related to the entire *model*, a particular *feature* it implements, or the way of its *execution*. The semantical meta information includes for example:

Model refinement level: The SAE J2546 standard proposes a number of model refinement levels, of which some are of interest in the context of a model library:

Pins: The model consists only of an interface with ports, parameters, and assertions. An instance of the model can be created, but cannot be executed since the model does not implement any internal feature.

Static: The model implements time-invariant, steady-state internal behaviour using some primary quantitative properties suitable for DC or steady-state AC analysis. On this level, an amplifier could be represented through a controlled source. A piezoresistive pressure sensor could be simply represented by its stiffness, stress-free resistance, and piezoresistive coefficient.

Dynamic: The model implements time-varying behaviour, possibly including nonlinear characteristics, but neglecting smaller second order effects to capture only the primary time-dependent behaviour. For example, the amplifier model could include its limited bandwidth and a slew rate. A pressure sensor model could include stress stiffening, inertia, and damping effects.

Precision: The model implements a significant amount of second order effects in addition to the effects necessary to capture its primary time-varying behaviour. For an amplifier model, this could mean that it includes noise and thermal effects like self-heating and temperature dependent gain variation. A pressure sensor model could include cross-sensitivity to other physical quantities like temperature as well as fatigue or hysteresis effects.

Vector: The model implements directional or spatial interfaces to its environment in contrast to the previously one-dimensional lumped connection points. This could be achieved for example through a distribution function or multiple connection points. Typical examples are models with spatially distributed parameters, e.g., a MOSFET device model taking into account its geometry and doping profiles or a 3-D FE model of a pressure sensor.

Model of Computation (MoC): on which the model is based. It may be a discrete MoC (DE, FSM, Petri net, DF, etc.), continuous MoC (signal flow, conservative network, bond graph, FEM, etc.), or an in some way synchronised mix of discrete and continuous MoCs.

Physical disciplines: the function of a modelled component is based on, e.g., electrical, mechanical, thermic, hydraulic, optic, ...

Validity of the model: describes under which assumptions and operating conditions a model is valid.

Suitability for design tasks: e.g., architecture exploration, area/power estimation, connectivity verification, or bottom-up verification (using back-annotated or calibrated models).

Keywords: to index the model.

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

Feature properties: A model has individual *features*, each one capturing a different aspect of the component with a varying level of detail: the *nominal behaviour* (e.g., amplification or signal filtering) of the component, plus *secondary behaviour* (e.g., temperature dependency or noise) caused by its interaction with the operation environment. A captured model feature can be mapped on the internal variables of a model, e.g., via global variables or the parameters and ports of the instantiated model.

A feature is often based on a *physical effect*, e.g., variation with T , manufacturing tolerances, ageing, self inductance, or noise. It influences the *performance criteria* of a component, e.g., filter characteristic, filter order, cut-off frequency, bandwidth, sample frequency, or the noise figure.

The *feature refinement level* describes how far a model feature is implemented (levels 0–7 in SAE J2546): *None* (not included), *Named* (acknowledged but unimplemented), *Fixed* (adjustment only through editing the model or a non-related parameter), *Index* (offers discrete choice of discrete values or modes), *Static* (accepts parameter value prior simulation run), *Dynamic* (adapting to internal conditions during simulation), *Mutual* (adapting to external influences during simulation), *Directional* (adapting to directional external influences during simulation).

Execution capabilities: Depending on its implementation, a model is suitable for different types of execution, notably *simulation* and *synthesis*. The results obtained from the model execution need to be characterised.

A model can support for *simulation* different *analysis types* like *Continuity* and *Loads Analysis*, *Nominal Analyses* (DC, transient, small signal, and stability analysis), *Stress Analysis*, *Perturbation* or *Sensitivity Analyses*, *Worst-Case Analyses*, *Failure Modes and Effects Analyses (FMEA)*, or *Sneak Circuit Analysis (SCA)*. For each analysis type, the model can give different results (e.g., current, voltage, temperature, force, power, failure), obtainable either dynamically during simulation or only after its completion. Each result can be in a different form like *Flag*, *Message*, *Scalar*, *Waveform*, or *Relation* (non-time series, describing in the form of, e.g., an equation or a table the interaction of two or more variables).

Synthesis transforms a model through an algorithm into another model. During top-down design, details are added in each synthesis step. For example, a programme implementing an *algorithm* (C, MATLAB®, VHDL, etc.) can be transformed into an RTL description (VHDL, Verilog, etc.), then into a *gate level* description (SPICE or Verilog netlist), and finally into a layout. During bottom-up design, models can be simplified through reduced-order modelling methods to make them suitable for simulation on higher levels of abstraction.

It is possible to include all this supplementary information into free-form description fields, but for large model collections, like they are intended for ModelLib, it is better to structure them as far as possible and to store them in an adapted data structure. This has to be done as general as possible because not all properties, which a designer might want to store for a model, are known in advance. The ability to store the semantical meta information in a structured way will enable the AMS designer to better manage the models for different levels of design abstraction of the same physical component.

A large collection of models also requires a more efficient access to the models, besides browsing, to support the designer in selecting the model with the right fidelity for reuse in the current design task at hand (e.g., architecture exploration, detailed component characterisation, system validation). To do that, the designer has to send queries to the model library. In simple cases, this means querying for a *model name*, *keywords*, and full text search in the description fields of the interfaces, architectures, etc. If this is

not sufficient, more complex queries should be possible, e.g., for certain properties of the model to be in a specific range (e.g., detail level, modelled effects, design language, component properties).

The use cases for browsing and querying of models showed which information has to be provided by the developer when submitting a new model to the library. First, the files containing the source code, test benches, test data, simulation results, and other documents of the model must be made accessible through Uniform Resource Locators (URLs), preferably by submitting them to the repository of a revision control system. This eases their further (cooperative) development by keeping track of the modification history of each file and by helping to resolve conflicts between parallel contributions to the same file. Then, the meta information about the model, test benches, etc., needs to be extracted from these files and entered into the library using structured input forms. This process can be partially automatised using similar techniques like the ones used by documentation generation tools [73, 153, 173]. During the further development of the model, the meta information needs to be continuously updated to keep it in sync with the changes made. The library supports the development process by providing a forum for discussing the model and jointly improving its implementation and documentation.

In the beginning, the newly submitted model is stored in one of the private model classes of the model developer, for which he can specify the access rights. To publish the model to the other users by including it into the official collection of the library, the developer has to contact one of the content managers. They will do a first review to evaluate if the model has reached a level of quality to be made public. This review includes checks if the model source code is following some suggested coding rules (e.g., no syntax errors, well structured and commented code, consistent naming scheme, ...) as well as if the supplied meta information and documentation is correct (consistent with the model source code) and complete enough (permitting understanding and usage of the model by a third party). The content manager gives feedback to the author until the requirements for the publication of the model are met. The new model is then sorted into the appropriate public model classes and announced to be available under certain license conditions. This formalised submission process shall insure a basic quality standard of the available models in the library. Subsequently, other authorised users can comment on the different parts of the model, can give a rating of the model, and can contribute to its further development by providing patches and documentation. Rating charts of the models may be implemented to motivate the developers through a sense of competition. The goal of these measures is to create an active community of model developers, which help and motivate each other while constantly improving the model collection.

3.3. Implementation of the ModellLib Prototype

Several software components have to be integrated to meet the requirements arising from the use cases presented in Section 3.2. To store the meta information about the models, a database is a core component of ModellLib. To manage the source files that contain the models and their accompanying document and allow their collaborative development, a revision control system is required. To discuss the models and collaborate on the improvement of their implementation and documentation, an open document development platform like a wiki is needed.

A running prototype of ModellLib [104, 106] has been developed, which implements the basic features of a model library. It uses several open source tools:

PostgreSQL [147]: to manage the database that stores the meta-information and wiki pages;

Subversion/WebSVN [7, 37, 166]: to handle the model repository;

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

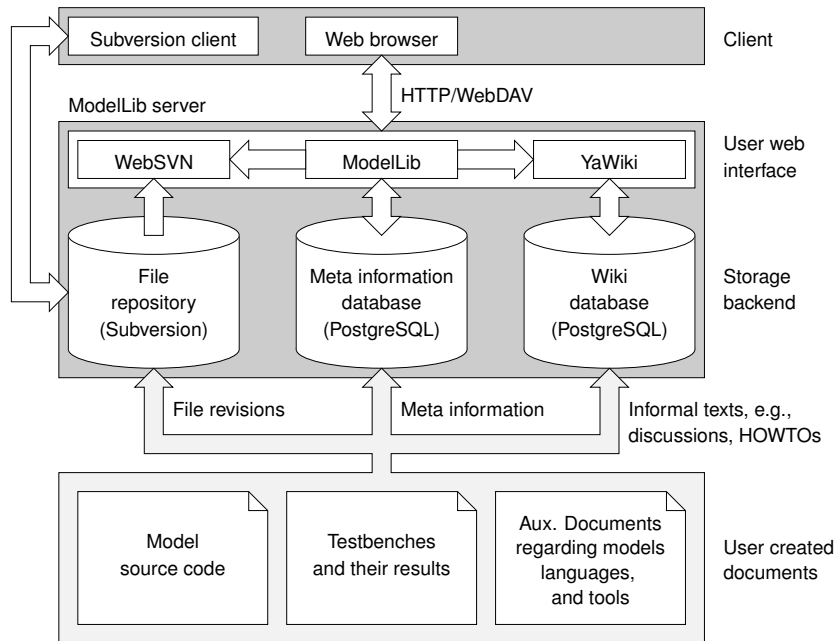


Figure 3.2.: Architecture of the ModelLib prototype depicting the communication links between its different components.

YaWiki [87]: to discuss and jointly develop the documentation of the models;

PHP [143]: to implement the web interface; and

Apache 2 [8]: to serve out the web interfaces of ModelLib, Subversion/WebSVN, and YaWiki.

Figure 3.2 shows the architecture of the prototype and how its different components interact. The lower part of the figure shows the different user-created documents managed by the ModelLib server. The file revisions of the documents are stored in the *repository*. The meta information about models and accompanying documents are stored in the *meta information database*. Informal texts, like discussions and HOWTOs, are stored in the *wiki database*. The user interfaces are implemented on the server side and provide the access to the data storages. This allows users to access ModelLib over the Internet using a standard web browser. On the user side, the file revisions of the models and documents are managed by the Subversion client. It provides the possibility to commit them to and update them from the repository.

The meta information describing the properties of the models is stored in a relational database. Figure 3.3 shows the Entity-Relationship (ER) diagram of the database that has been designed for the prototype. It currently considers the model class hierarchy, the information about referenced external documents, the information about available design language and tool versions, and the meta information about the interface, architectures, assertions, design entities, test benches, and files of the models. This fully covers the structural meta information described in Section 3.2. The semantical meta information is not yet implemented in the database structure and can be stored for the moment only as free-form descriptions. The Relational Database Management System (RDBMS) PostgreSQL [147] has been chosen to implement the database for the ModelLib project, among other reasons, because of its full ACID compliance, good ISO SQL standard coverage, and implementation of foreign key constraints.

The ModelLib web interface provides access to the model collection over the Internet. It queries the

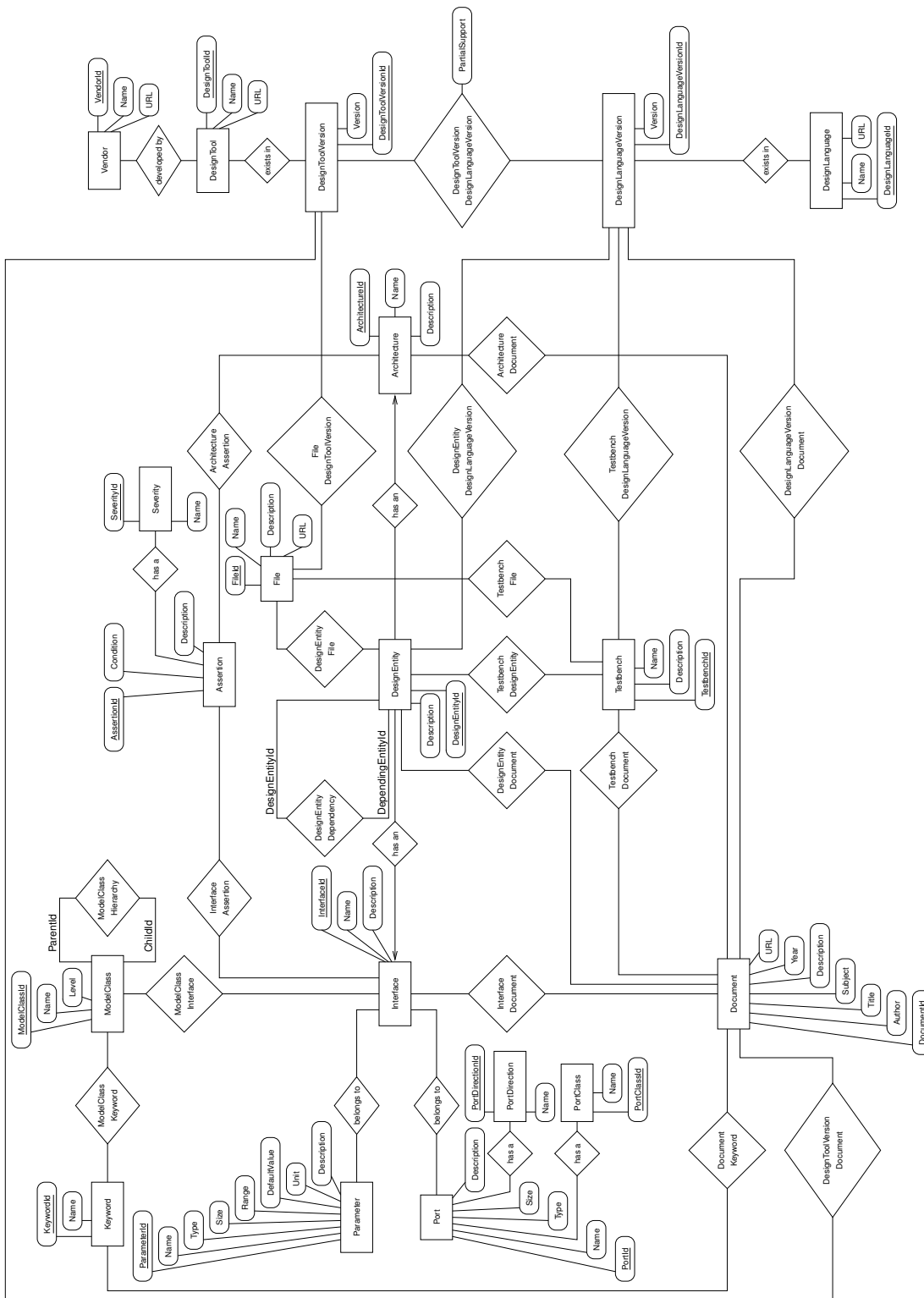


Figure 3.3.: Entity-Relationship diagram of the meta information database.

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

meta information about models, test benches, design languages, design tools, and document references from the RDBMS and outputs it to a CSS formatted HTML page. In the prototype, it is implemented in the server-side scripting language PHP using library packages from the public PEAR repository [144], which ease the development of web applications. For example, the implementation uses DB and Text_Wiki to access the database on the PostgreSQL server and format the HTML output of the free-form description fields using a wiki-like syntax. Currently, the web interface implements the following features:

- Logging in as different database users;
- Browsing the model class hierarchy for a model;
- Displaying and editing all information related to the interface, architectures, design entities, and test benches of a model (Figure 3.4);
- Displaying the information about the available design languages and their versions (Figure 3.5);
- Displaying the information about the available design tools and their versions (Figure 3.6); and
- Displaying/editing the document references.

The files containing the models, test benches, and accompanying documents are stored in a repository, which is managed by a revision control system to allow their collaborative development. The meta information database refers to the files in the repository using URLs.

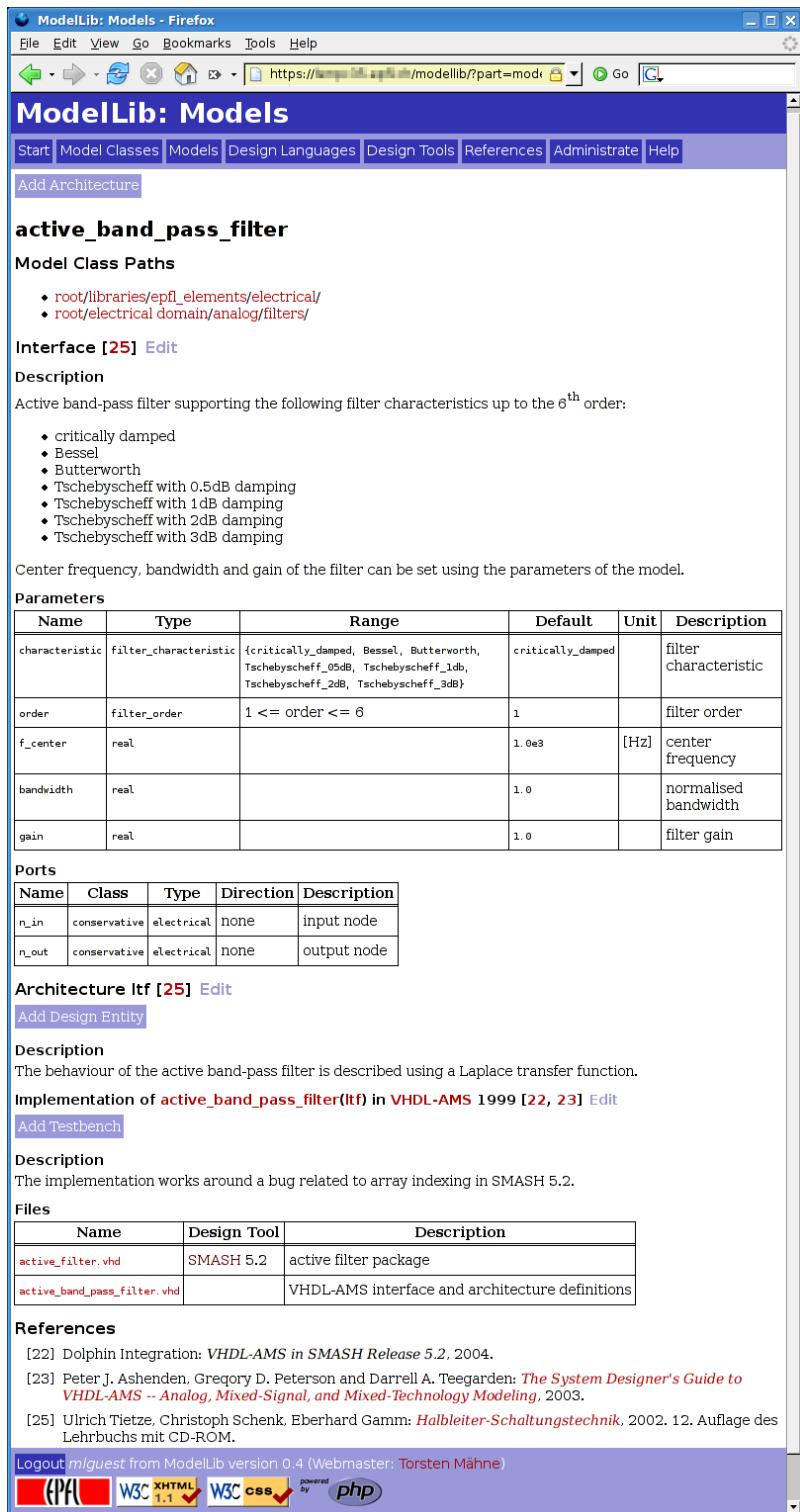
The *Subversion* system [7, 37] has been chosen for the ModelLib prototype because it has seen widespread adoption since it start in 2000, replacing the previously popular CVS [148]. It overcomes some of CVS's known drawbacks by supporting, among others, versioned directories, renames, and meta data; truly atomic commits; efficient handling of binary files; and more efficient handling of tags, branches, and merging. At the same time, its command line interface remained simple and comfortable. GUI front ends and IDE integrations are available. Using the module `mod_dav_svn.so` from Subversion, an Apache 2 web server makes the repository available to the clients via the WebDAV protocol [161], which is an extension to the HTTP 1.1 protocol that adds versioned writing capabilities. This provides key features like authentication, path-based authorisation, wire compression, and basic repository browsing using a web browser or WebDAV client. The PHP-based web interface *WebSVN* [166] considerably improves the browsing of the repository via a web browser with features like viewing the file/directory logs; listing of all changed, added, or deleted files in a queried revision; log message searching; blame support; Tar ball downloads; directory comparisons; and RSS feed support.

A wiki provides an open platform to discuss the models and to collaborate on their implementation and documentation. An access rights management has to be established to control the read and write access to the different models. *YaWiki* [87] was chosen for the ModelLib prototype because it adds logical name spaces, Access Control Lists (ACLs), navigational elements, and more to a traditional wiki; each wiki page can be instantly commented through a web form; it is written in PHP like the other parts of the ModelLib web interface; its formatting engine *Text_Wiki* is a PEAR module, which is also used in the ModelLib web interface to format the free-form description fields.

3.4. Fine-Grained Access Control Mechanism for the Meta Information

One important aspect for the success of a model library is the management of the IP represented by the models. A *fine-grained access control mechanism* is thus necessary to disclose selectively information

3.4. Fine-Grained Access Control Mechanism for the Meta Information



ModelLib: Models

Start | Model Classes | Models | Design Languages | Design Tools | References | Administrate | Help

[Add Architecture](#)

active_band_pass_filter

Model Class Paths

- root/libraries/epfl_elements/electrical/
- root/electrical domain/analog/filters/

Interface [25] Edit

Description

Active band-pass filter supporting the following filter characteristics up to the 6th order:

- critically damped
- Bessel
- Butterworth
- Tschebyscheff with 0.5dB damping
- Tschebyscheff with 1dB damping
- Tschebyscheff with 2dB damping
- Tschebyscheff with 3dB damping

Center frequency, bandwidth and gain of the filter can be set using the parameters of the model.

Parameters

Name	Type	Range	Default	Unit	Description
characteristic	filter_characteristic	{critically_damped, Bessel, Butterworth, Tschebyscheff_05dB, Tschebyscheff_1dB, Tschebyscheff_2dB, Tschebyscheff_3dB}	critically_damped		filter characteristic
order	filter_order	1 <= order <= 6	1		filter order
f_center	real		1.0e3	[Hz]	center frequency
bandwidth	real		1.0		normalised bandwidth
gain	real		1.0		filter gain

Ports

Name	Class	Type	Direction	Description
n_in	conservative	electrical	none	input node
n_out	conservative	electrical	none	output node

Architecture Itf [25] Edit

[Add Design Entity](#)

Description

The behaviour of the active band-pass filter is described using a Laplace transfer function.

Implementation of active_band_pass_filter(Itf) in VHDL-AMS 1999 [22, 23] Edit

[Add Testbench](#)

Description

The implementation works around a bug related to array indexing in SMASH 5.2.

Files

Name	Design Tool	Description
active_filter.vhd	SMASH 5.2	active filter package
active_band_pass_filter.vhd		VHDL-AMS interface and architecture definitions

References

- [22] Dolphin Integration: *VHDL-AMS in SMASH Release 5.2*, 2004.
- [23] Peter J. Ashenden, Gregory D. Peterson and Darrell A. Teegarden: *The System Designer's Guide to VHDL-AMS -- Analog, Mixed-Signal, and Mixed-Technology Modeling*, 2003.
- [25] Ulrich Tietze, Christoph Schenk, Eberhard Gamm: *Halbleiter-Schaltungstechnik*, 2002. 12. Auflage des Lehrbuchs mit CD-ROM.

Logout [m/guest](#) from ModelLib version 0.4 (Webmaster: Torsten Mähne)










Figure 3.4.: Meta information about a selected model.

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

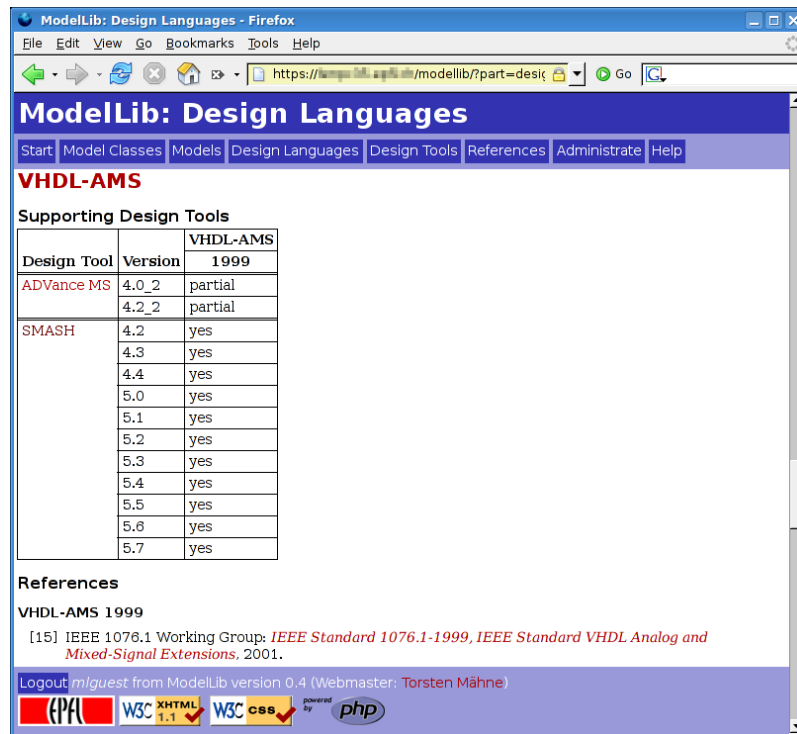


Figure 3.5.: Meta information about design languages.

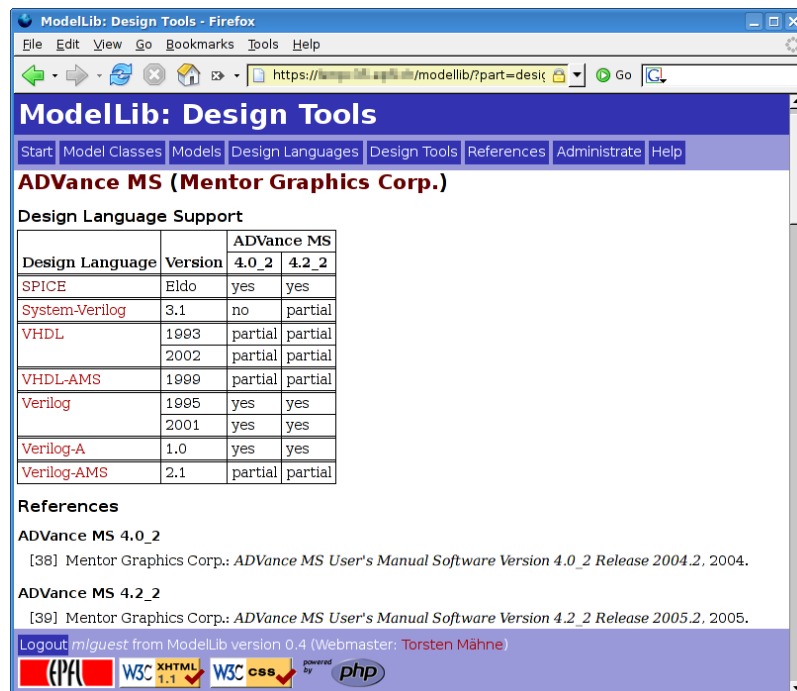


Figure 3.6.: Meta information about design tools.

to different user groups. For example, the public might only have access to high-level models of a component or only the full interface information plus a compiled/encrypted model, whereas a customer of the design company has full read access to the documentation and the model sources, and the model developers themselves have full read/write access to all information.

Subversion and YaWiki already implement mechanisms to limit the access to files/wiki pages with respect to the user and its group memberships using ACLs. However, the real key to the models in the repository and the documents in the wiki is the meta information about the models stored in the relational database and thus may contain sensitive information. The RDBMS PostgreSQL provides authentication and access control mechanisms through its *role concept* and the *grant/deny of privileges* on tables, views, and function execution. Each role can be member of other roles permitting the common definition of privileges. Unfortunately, the definition of privileges on the table level is too coarse in the case of the model library, because information common to all models, e.g., about the interface, are stored in the same table. For this reason, a tuple-wise (per table row) access control mechanism has been implemented within the database using only the features provided by the RDBMS in the scope of Thomas Böhm's master's thesis [21] under supervision of the author of this Ph.D. thesis.

In the chosen approach, which is illustrated in Figure 3.7, the *Select*, *Update*, *Delete*, and *Insert* rights on the tables are granted only to the administrators (members of role *mladmin*). To the authorised users and guests (members of *mluser* and *mlguest*) is granted only the *Select* right on views, to which the rows from the underlying tables are propagated selectively according to the entries in an additional ACL table for the roles the database user is member of. The authorised users have also the *Update*, *Delete*, and *Insert* rights granted on the views. The modification of the view data via *Update*, *Insert*, or *Delete* actions are handled by rewriting rules [146]. These rules are triggered by the mentioned actions and only propagate actions to the underlying tables that are permitted by the entries of the attached ACL. Not permitted modifications are dropped.

The rights administration would be too tedious, if the rights were specified for each single tuple. That is why a rights inheritance scheme (Figure 3.8) has been implemented into the view queries and rules allowing to reuse the rights defined on entities higher in the hierarchy and permitting their overriding through the local ACL. The tables with names in bold have an own ACL, which is merged with the inherited rights definitions. The entities **Documents** and **Files** are not inheriting access rights from other instances because they can be shared between several models.

The implementation on the database layer has the advantage that it simplifies the development of the higher layers (application logic and presentation) and that it allows the reuse of the access control implementation for different interfaces (e.g., web interface and EDA tool links).

3.5. Towards a 3-Tier Reimplementation of ModelLib

With the access control mechanism for the model meta information in place, the attention returned to the ModelLib prototype itself. The first prototype had served well to demonstrate and validate the ideas for a model library infrastructure. However, the experiences from its development had shown weaknesses in its technological foundation, which would complicate the implementation of the remaining requirements, especially the implementation of an API for EDA tools to access ModelLib and the consideration of semantical meta information and complex queries on the meta information database. The usage of PHP for the web interface development allowed a rapid prototyping of the ModelLib user interface, but did not allow to cleanly separate the application logic from the presentation layer. Therefore, it was decided to reimplement the web interface to improve the modularity of the data storage, application logic, and

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

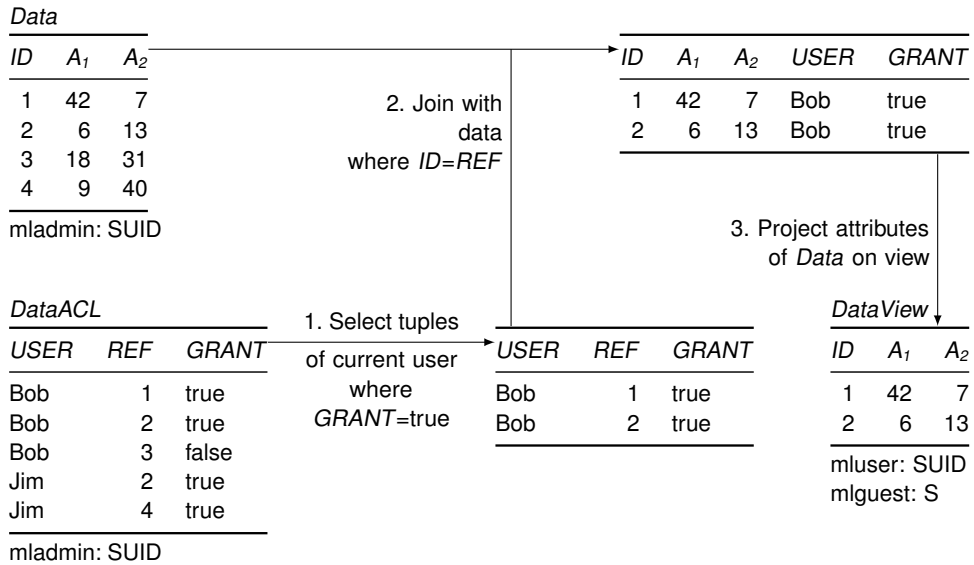


Figure 3.7.: Implementation of the tuple-wise access control mechanism. Below each table, the access rights to it for the different database roles are listed: (S)elect, (U)pdate, (I)nsert, and (D)elele. Rewriting rules handle the Update, Insert, and Delete actions on the view by forwarding permitted actions to the underlying table and dropping not permitted actions.

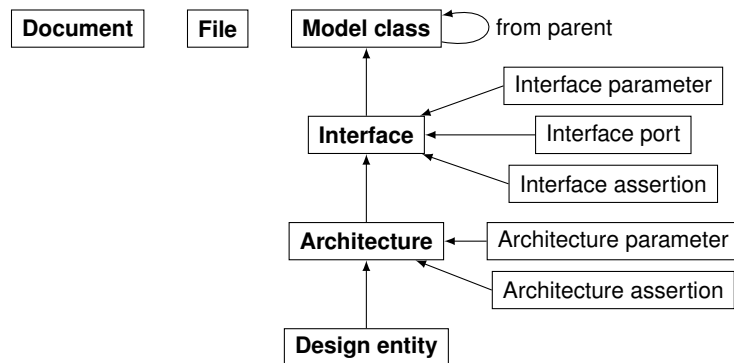


Figure 3.8.: Access rights inheritance between the tables storing the model meta information. The tables with names in bold have an own ACL, which is merged with the inherited rights definitions. The entities Documents and Files are not inheriting from other instances because they can be shared between several models.

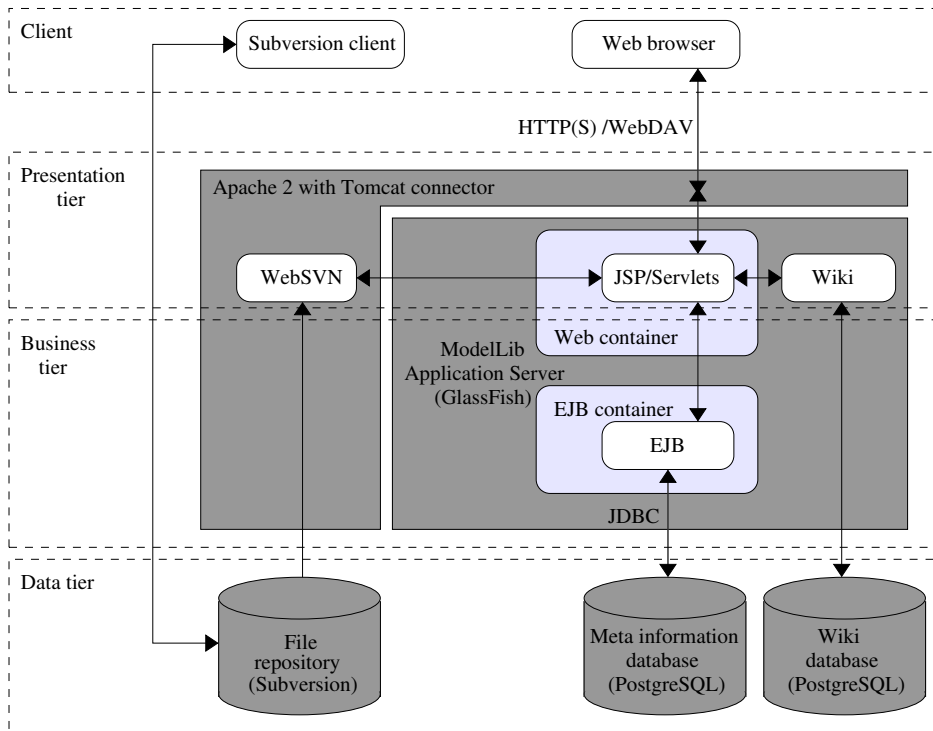


Figure 3.9.: Proposed architecture for the Java EE reimplementation of ModelLib [21].

presentation layers. Thus, a future API for EDA tools to directly access ModelLib would not require a duplicated implementation of the ModelLib application logic. This work was started in the scope of two master's thesis projects [21, 177], which were supervised by the author of this Ph.D. thesis.

Thomas Böhm [21] evaluated the available technology options and proposed Java EE 5 [85] as the new implementation platform for the next-generation ModelLib prototype. He made an initial proposal for a 3-tier architecture, which is shown in Figure 3.9. The PostgreSQL database and Subversion repository would be kept as well as the Apache 2 web server, which would be augmented with a Tomcat connector to interact with the added GlassFish application server. The new Java EE-based ModelLib application would be deployed on the application server. Its business tier implementing the access to the meta information database and the ModelLib application logic would run within an EJB container. The application tier would run in an independent Web container and consist of JavaServer Pages (JSPs) and servlets implementing the web user interface of ModelLib. A future EDA API for ModelLib would run in yet another container (not shown in Figure 3.9), which would directly interact with the application tier. The YaWiki of the original prototype would be replaced by a Java-based pendant also running on the application server.

Daniel von dem Knesebeck [177] refined this initial architecture proposal (Figure 3.10) and started the implementation of the Java EE-based new ModelLib prototype. Within the time constraints for his master's project, he managed to implement the login/session management component for the web interface, the user and access control management component for ModelLib, and the data persistence layers, which is responsible for the object-relational mapping of the meta information database onto corresponding Java objects. In Figure 3.10, these components are highlighted in grey. The implementation of the data persistence layer was complicated due to a limitation of the used NetBeans Integrated

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

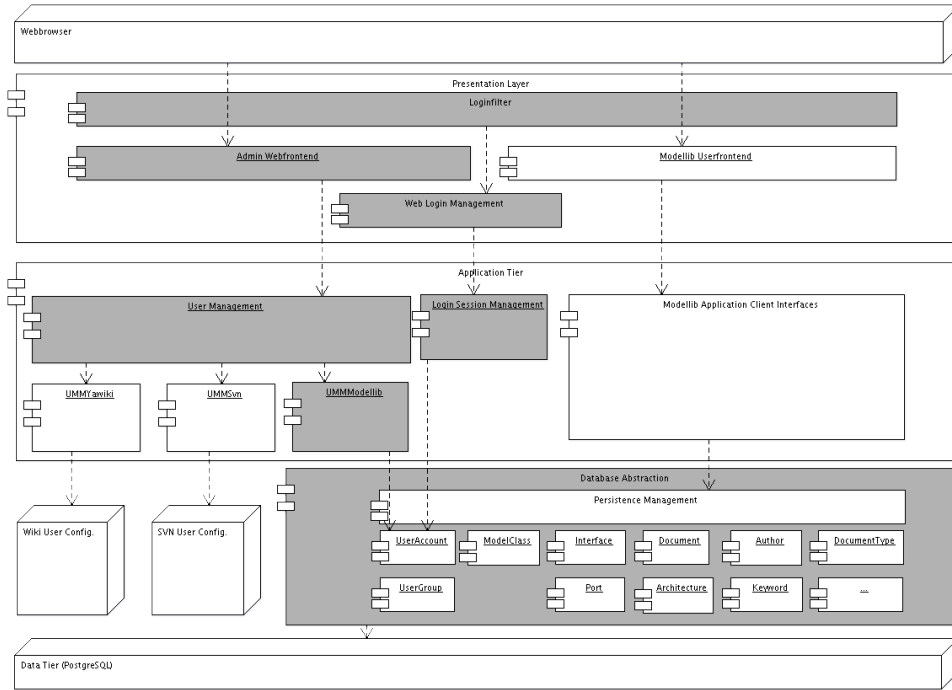


Figure 3.10.: Refined architecture for the Java EE reimplement of ModelLib [177]. Only the components highlighted in grey have been implemented in the cited work.

Development Environment (IDE), which could at that time automatically import tables but not views from the PostgreSQL database. Therefore, the access to the views, which are central to the implemented tuple-wise access control mechanism (Section 3.4), needed to be implemented manually. This also meant that small changes to the meta information data model, which are normal in this early stage of the project, required many additional changes in the access control and persistence layer to be taken into account. Therefore, the application logic and user interface of the ModelLib application could not be implemented anymore. The new Java EE architecture proved to be somewhat “heavy” in the light of the incremental prototype development of ModelLib. To resolve these issues on the computer science side, more development resources than were available for this project would have been needed. Since these problems were clearly outside the scope of this Ph.D. thesis, it was decided to not pursue the ModelLib development further than the described point.

3.6. Conclusions and Outlook

This chapter presented the concepts for a web-based platform to foster the reuse and collaborative development of models in the AMS SoC Design Process, which provides central place to collect and manage models and their associated meta information. The developed ModelLib prototype presented in Section 3.3 implements the basic features of such a model library as described in Section 3.2. Its web interface allows browsing for a model through the model class hierarchy. The meta information about models, test benches, design languages, design tools, and external documents that are stored in the meta information database can be displayed. New models can be added by committing them into the repository and adding/editing their meta information in the meta information database using the

web interface. A fine-grained access control mechanism has been implemented on the database layer to allow the public usage of ModelLib over the Internet while keeping control on who has read and write access to the different parts of the library. The querying of models is currently only supported through direct SQL queries to the meta information database. This use case needs to be implemented into the web interface, so that it will offer the designer elaborated query schemes to guide him/her in the selection of a suitable model for the current design task at hand.

The presentation of the running ModelLib prototype in the scope of FDL 2006 [105] and PRIME 2007 [107] yielded external interest in the project and gave input in the form of requirements to consider additional kinds of meta information and to enable a tighter tool integration. One idea was to automatise the import of models and their meta information using a modified documentation extraction tool [153] and a meta information exchange format, which needs to be standardised. Another idea was to augment the collected models with synthesis-related information to qualify them as soft-IP input for the hierarchical synthesis of AMS systems [128]. Besides an extensible meta information model, this would also require a refined API for the EDA tools. They would not only provide a user interface to the model collection stored in ModelLib, but would autonomously select models with certain properties and store models of firm-IP (i.e., qualified data points for a given technology and simulator) generated during iterative synthesis processes.

These research perspectives and the identified unsatisfying separation of the application logic and presentation aspects of the PHP-based ModelLib web interface led to the decision to start a reimplementation of the ModelLib application using the feature-rich Java EE platform. The goal was to improve the modularity of the data storage, application logic, and presentation layers and to realise direct EDA tool access through an API. This new technological foundation should have simplified the implementation of usability improvements and of missing functionality in the web interface. It would have included work towards a better integration of the three main components, i.e., repository, meta information database, and wiki to offer the user a coherent interface. The usage of Java EE indeed allowed a better modularisation of the ModelLib application, but unfortunately proved also to be a burden on the prototype development. Java EE's feature richness greatly simplifies the implementation of certain aspects (e.g., the implementation of the data persistence layer or login/web session management). However, it also requires an evaluation/understanding of the many Java/web technologies and their interaction to make the right choices for the ModelLib prototype development. Thus, only the login/web session, user/access control management, and data persistence layer components could be finished with the limited available development resources.

Another complication for the ModelLib prototype development was the evolving data model for the model meta information. Even though the presented ER diagram for the meta information database supports well the storage of the structural meta information about the model, the prototype development and usage revealed smaller aspects, which needed to be improved. This usually implied only the modification/addition of attributes to an entity. Unfortunately, each of these changes needed to be manually propagated to the access control and data persistence layers of the ModelLib prototype due to a lacking tool support for the import of database views by the used IDE at the time when the presented work was done. This would have applied also to the further development of the ModelLib prototype, in which the main task would have been to support the structured storage of the semantical meta information. As with the abovementioned research perspectives, this would have implied serious extensions of the current database scheme. Therefore, a way needs to be found to incrementally develop the ModelLib prototype and the underlying meta information database to allow successive implementation of the different model library use cases and the evaluation of new ideas. Ideally, the meta information data model should be modularised and clearly defined in a kind of XML schema. The database schema, the configuration

3. ModelLib: A Web-Based Platform for Collecting Behavioural Models

of the access control mechanism, the object-relational mapping layer for the web application, and the model meta information exchange file format could then be automatically derived from that data model definition schema. However, the development of the required tools is outside the scope of this thesis and was well beyond the available resources.

Still, the development of ModelLib was a valuable experience. The use cases developed for this tool show how it can support the AMS SoC design process by fostering the reuse and collaborative development of models for tasks like architecture exploration, system validation, and creation of more and more elaborated models of the system. The experiences from the ModelLib development delivered insight into which aspects need to be especially addressed throughout the development of models to make them reusable: mainly flexibility, documentation, and validation. This was the starting point for the development of an efficient modelling methodology for the top-down design and bottom-up verification of RF Systems based on the systematic usage of behavioural models which is described in the Chapter 4.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems for their Design and Verification

This chapter presents a modelling methodology for the top-down/bottom-up design of RF systems based on the systematic use of VHDL-AMS models. The model interfaces are parameterisable and pin-accurate. The designer can choose to parameterise the models using performance specifications or device parameters back-annotated from the transistor level implementation. The abstraction level used for the description of the respective analogue/digital component behaviour has been chosen to achieve a good trade-off between accuracy, fidelity, and simulation performance. These properties make the models suitable for different design tasks such as architectural exploration or overall system validation. This is demonstrated on a model of a binary Frequency-Shift Keying (FSK) transmitter parameterised to meet very different target specifications. The achieved flexibility and systematic documentation of the models facilitate their reuse in other design projects.

4.1. Introduction

Portable, battery-powered electronic devices (cell phones, PDAs, notebooks, mice, ...) nowadays routinely contain wireless functionality to communicate among each other using standards like Bluetooth™, Wi-Fi™, or GSM™. This became possible due to the integration of full RF transceivers into a SoC, which are built of these main parts (Figure 4.1):

Frequency synthesiser: based on a Phase-Locked Loop (PLL) consisting of Voltage Controlled Oscillator (VCO), DIViders (DIVs), Phase/Frequency Detector (PFD), Charge Pump (CP), Loop Filter (LPF), Quartz Oscillator (QO) for the reference frequency, Σ - Δ modulator for fractional frequency division and Multi-stage-noise-SHaping (MASH) to generate the RF carrier signal required for modulation and demodulation.

Receiver (RX) chain: consisting of Channel Selector (CS), Low-Noise Amplifier (LNA), MIXer (MIX), Intermediate Frequency Amplifier (IFA), Poly-Phase Filter (PPF), and DSP.

Transmitter (TX) part: modulating the Pulse-Shaped (PS) data bit stream on the RF carrier using direct modulation via the fractional PLL and amplifying the RF signal with a PA.

POWER management (POW): including linear regulators and step-up converters.

Digital ConTRoL (CTRL) of the transceiver through some communication interface (e.g., Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I²C) bus) or any other digital system (e.g., a microcontroller).

The design of such a complex system is highly demanding since its analogue/RF and digital parts require different design flows, methodologies, and tools.

During the *top-down design phase*, the overall system specifications must be properly distributed and assigned to the components defined by the selected architecture. Then, the result has to be evaluated and

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

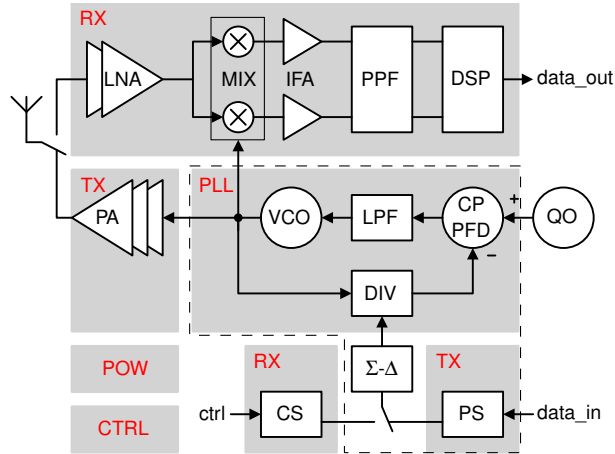


Figure 4.1.: Architecture of an RF transceiver for frequency-modulated signals.

checked against constraints such as the feasibility with the selected technological process. The distributed specifications will then be used for the detailed block design until their layout is obtained. While HDLs such as VHDL or Verilog are routinely used for the design of digital parts, mathematical tools such as MATLAB®/Simulink® or spreadsheets are still preferred for deriving analogue/RF specifications and performing system level studies such as link budget analysis. However, mixed-signal HDLs such as VHDL-AMS or Verilog-AMS offer additional capabilities that can support the top-down design and more specifically the architectural exploration step. Anticipating the real component behaviours with models at proper levels of abstraction using appropriate parameters is a considerable help in that phase. Their reusability for evaluating various possible architectures is also an advantage [63].

During the *bottom-up verification phase*, the implemented components must be individually verified and then assembled to form the complete system under design. Assuming that each component has been successfully verified, a full top level verification is necessary for detecting interface issues such as wrong signal behaviour or incorrect timing. As full system verification at transistor level is becoming impracticable due to the huge amount of data to handle, models at proper levels of abstraction using appropriate parameters again offer the required accuracy/verification performance trade-off [151].

This chapter reports on the development of a library of VHDL-AMS models called RF_TRX applying a dedicated modelling methodology. The goal is to support both the top-down architectural exploration and the bottom-up verification of complex RF systems such as the presented RF transceiver. The starting point was an existing design of an FSK transmitter (indicated by the dashed line in Figure 4.1) provided by the RF group of the Centre Suisse d'Électronique et de Microtechnique SA (CSEM). From its circuit topology, behavioural VHDL-AMS models of each of its components have been abstracted. The interfaces of each component model were generalised and their behaviour parameterised using a particular modelling methodology, which is presented in Section 4.2. Special care has been taken to model the proper non-idealities, while maintaining acceptable simulation performances, and to define consistent parameter sets that support the top-down (specification) and bottom-up (back-annotation) phases. Similar model libraries have been already presented, e.g., in Milet-Lewis et al. [120], Frevert, Haase, and Jancke [54], and MGC [117], but they mostly focus on top-down design only.

The chapter is organised as follows. Section 4.2 presents the modelling methodology that has been used for the modelling of the frequency synthesiser and its components described in Section 4.3. The development of a model of the VCO is described in detail from the designer's specification to the actual

VHDL-AMS model as well as its validation using an elaborate test bench and transistor level reference model. Other component models needed to model the frequency synthesiser are only outlined as they have been specified and developed in the same way. Section 4.4 presents the usage of the developed VHDL-AMS models for the architectural exploration of an FSK transmitter in two different design cases using carrier frequencies of 868 MHz and 2.45 GHz. The section also shows how the developed structural transmitter model can be reused during bottom-up verification of transistor level implementations of individual components and how modelling decisions impact the simulation performance. Finally, Section 4.5 draws some conclusions and gives an outlook on future steps.

4.2. Modelling Methodology

In this section, the modelling methodology is outlined. Its application is illustrated in Section 4.3 using the frequency synthesiser and particularly its VCO component as an example.

The first step involves the *definition of the requirements the model library shall meet*. This includes:

- The definition of the components that will be modelled,
- The level(s) of detail, at which the component behaviours shall be described,
- The design tasks (e.g., top-down architecture exploration, bottom-up verification) that the component model shall support,
- The parameters that shall be available to the model user (primary parameters), and
- The parameters that shall be derived internally (secondary parameters).

The definition of these two kinds of parameters has to be done carefully as the same model shall be used for both top-down architecture exploration and bottom-up verification. In the top-down phase, the interface parameters (in the context of VHDL-AMS also known as *generic parameters* or just *generics*) are used as specifications and internal parameters are used as nominal values for the design of the component at the transistor level. In the bottom-up phase, the interface parameters are back-annotated with values from the transistor level implementation and internal parameters have in turn values that correspond to extracted specifications. This also means that models shall be pin-accurate for easy replacement with equivalent transistor level models, when required.

The second step involves the *specification of the behaviours to model, their interfaces (signals and parameters), and their internal parameters* as discussed above. This depends largely on which specifications need to be validated with the help of the (sub-)system model during the different design tasks. It requires a preliminary understanding of the overall system function to derive the principal component parameters that have an influence on the system performance. Only non-ideal behaviours should be considered in the component models, which have a direct impact on the system design and can be approximately quantified or described without retaining all the details of the transistor level implementation. Otherwise, there will be no advantage of using a behavioural model rather than the transistor level model of a component. The specification is largely prepared by the domain experts, i.e., in our case the RF designers, with feedback given by the model developers, e.g., asking for clarification or restructuring of certain specifications.

Then, the *agreed specifications have to be translated into legal VHDL-AMS models*. In this step, the capabilities offered by the modelling language and, unfortunately, the incomplete support of the

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

language in existing EDA tools, have to be balanced to achieve the intended accuracy, fidelity and simulation performance. For example, applying event-driven modelling techniques when describing analogue behaviours may make a model more robust and faster to simulate. For this work, it has been decided to define only one **entity** per component with a given port interface. This entity includes all specified primary parameters for the different design tasks. This facilitates the reuse of structural models throughout the top-down/bottom-up design process. The model behaviour is implemented in one or more **architectures**, each targeting certain design tasks with its specified level of detail. Each architecture may use only a subset of the generic parameters specified by the entity and calculate from them its internal parameters. The implemented behaviour is expressed in terms of the signals/quantities bound to the entity's ports/terminals, the internal signals/quantities, and the generic/internal parameters. If two design tasks require the same level of detail (i.e., the same behaviour) but different generic parameter sets (e.g., performance specifications for the top-down design or transistor level device parameters for the bottom-up verification), it can be beneficial to derive from both generic parameter sets a common set of internal parameters that are then exclusively used to express the model's behaviour. Thus, the code implementing the behaviour can be largely reused. Only the initialisation of the internal parameters in the architecture will differ as well as maybe some architecture level assertions. Supplementary internal parameters might also be calculated, which are unrelated to the primary behaviour of the model but are valuable for the RF designer. For each entity, an identical **component** definition is available in a package. The designer is encouraged to use exclusively these components in his structural models. Thus, he can conveniently select for each component instance the architecture to be used during a simulation for a certain design task by means of an external **configuration**.

Finally, the *models shall be thoroughly tested using elaborate test benches* that include as much as possible the verification of assertions about the modelled components. If reference models or measurements of these components exist, the new generalised model shall be validated against them.

As these steps imply a non-negligible implementation effort, the models and test benches shall be as flexible and modular as possible to be *reusable in other design projects*. In this way, the design process can be accelerated and optimised using an over time growing base of well documented and individually reviewed/verified component models, in which validity other designers can trust. The particular requirements for such a model library and an infrastructure supporting it have been described in detail in Chapter 3. The model implementations need to be well documented for facilitating the maintenance of the code and possible addition of further component effects. This also requires the development of (automated) test benches allowing to check as far as possible that modifications don't break the model itself or dependent models.

4.3. Modelling the Frequency Synthesiser

4.3.1. Specification of the Frequency Synthesiser Behaviour

A frequency synthesiser [152] is a system that allows to enslave an oscillator at a specific frequency. It is most of the time based on a PLL (Figure 4.1) that includes a VCO. For the RF application described in this chapter, the VCO is designed to give a signal at high frequencies, like those specified for the Industrial, Scientific, Medical (ISM) or Short Range Device (SRD) bands. Such oscillator should moreover have a low phase noise, but this point will not be treated or modelled here (see Hajimiri and Lee [69] for more informations on phase noise in oscillators). The frequency divider counts the edges of the signal coming from the VCO and gives a digital rising edge to the PFD when the count has reached the dividing value given by the N_{div} input. As long as the edges coming from the frequency divider

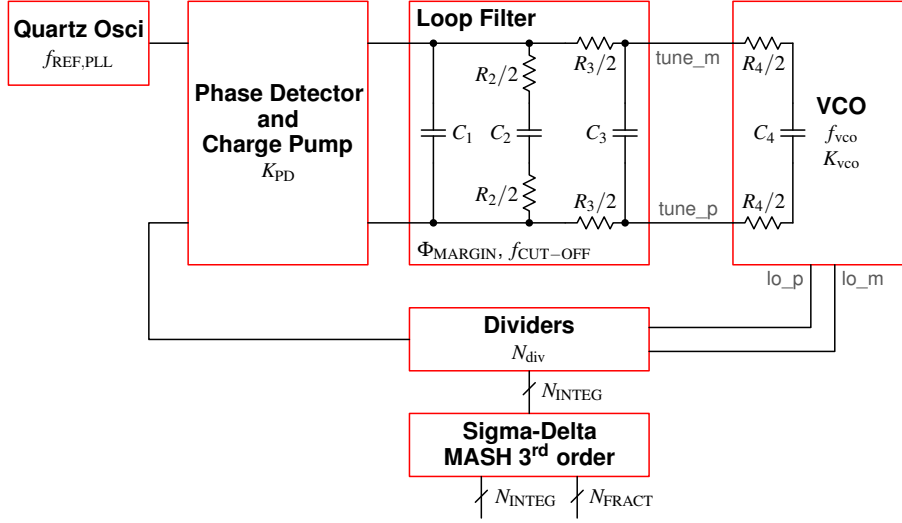


Figure 4.2.: Global structure of the frequency synthesiser.

and the quartz oscillator are not in phase, the PFD will order the CP to increase or decrease the tuning voltage, filtered by the LPF, at the input of the VCO. Thus, the frequency of the VCO will change until its frequency, divided by the ratios of the frequency dividers, is exactly equal to the reference frequency supplied by the quartz oscillator. The equation expressing the relation between VCO frequency f_{vco} , the division ratio N_{div} , and the quartz reference frequency $f_{REF,PLL}$ is:

$$f_{vco} = f_{REF,PLL} \cdot N_{div} \quad (4.1)$$

Please also note in this equation a convention used for the rest of this chapter: Variables, which stay constant throughout the whole simulation, i.e., which are to be passed as generics or to be calculated once during model initialisation, are denoted with indices in upper case. Variables changing over the course of the simulation have lower-case indices. This convention helps later to decide which variables need to be declared as constant and which as quantities or signals during the design and implementation of the model.

As for every looped system, a *stability problem* occurs if the phase margin of the PLL open-loop gain is too small. For calculating this open-loop gain, we need to define the principal parameters (Figure 4.2) of each block:

VCO: K_{vco} in Hz/V, expresses the relationship between the effective input tuning voltage $v_{tune,eff}$ (in Figure 4.2 the voltage over C_4) and the output VCO frequency f_{vco} :

$$f_{vco} = K_{vco} \cdot v_{tune,eff} + f_{vco,0} \quad (4.2)$$

with $f_{vco,0}$ being the free running frequency of the VCO for $v_{tune,eff} = 0$ V.

Frequency divider: N_{div} , modulated by the Σ - Δ modulator, is the division ratio between the VCO and the quartz oscillator frequencies. In an implementation, N_{div} might be obtained by cascading several (programmable) dividers and controlling their individual division ratios with a dedicated control block.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

PFD and CP: K_{PD} in A/rad, expresses the gain between the phase error at the input of the PFD and the instantaneous current flowing at the output of the CP.

LPF: C_1, R_2, C_2, R_3 , and C_3 are the circuit device parameters of the loop filter, which need to be adjusted to achieve a proper phase margin Φ_{MARGIN} for a specific cut-off frequency $f_{\text{CUT-OFF}}$. Note that a fourth order R_4, C_4 (impedance at the VCO input) may not be negligible and has therefore to be taken into account. Depending on the VCO input impedance, also a loop filter of lower order (omitting R_3 and C_3) might be sufficient.

Using these parameters, the open-loop transfer function of the PLL is given by the following equation:

$$H(j\omega) = \frac{\frac{K_{VCO} \cdot K_{PD}}{N_{DIV} \cdot j\omega} \cdot Z(j\omega)}{(1 + j\omega \cdot R_3 \cdot (C_3 + \frac{C_4}{1 + j\omega \cdot R_4 C_4})) \cdot (1 + j\omega \cdot R_4 C_4)} \quad (4.3)$$

where $Z(j\omega)$ is the impedance seen by the charge pump:

$$Z(j\omega) = \frac{1}{j\omega \cdot \left(C_1 + \frac{C_2}{1 + j\omega \cdot R_2 C_2} + \frac{C_3 + \frac{C_4}{1 + j\omega \cdot R_4 C_4}}{1 + j\omega \cdot R_3 \cdot (C_3 + \frac{C_4}{1 + j\omega \cdot R_4 C_4})} \right)} \quad (4.4)$$

By calculating the phase value, for which the magnitude of Eq. (4.3) is equal to one, the *system phase margin* can be derived [97]. In the next section, the behaviour of the VCO is specified in more detail. For all other component models, the same approach has been used.

4.3.2. Specification of the Voltage Controlled Oscillator Behaviour

The task of an oscillator is to generate a periodic signal at a certain frequency with well known properties. Different implementations can be used to build the VCO by satisfying the gain and phase conditions needed to ensure steady-state oscillation [101]. When used in a frequency synthesiser, the oscillator has to cover a precise output frequency range determined by the target application. Thus it has to be controlled somehow in an analogue way (e.g., by a voltage supplied to a varicap) and, optionally, in a digital way (e.g., by using a switched capacitor bank). A tunable ideal oscillator with continuous output phase and control voltage $v_{\text{tune,eff}}$ can be represented by the following equation:

$$v_{\text{osc}}(t) = V_{\text{AMP}} \cdot \cos\left(2\pi\left(K_{VCO} \int_{-\infty}^t v_{\text{tune,eff}} dt + f_{VCO,0} \cdot t\right)\right) \quad (4.5)$$

For this work, an *LC* oscillator (Figure 4.3) with band switching capabilities is modelled. It is composed by an inductor L_{TANK} and several capacitors, which fix together the oscillator frequency f_{VCO} , while some active devices compensate for the losses in order to sustain the oscillation. The frequency tuning is performed by adjusting the capacitance value using an analogue and a digital control in parallel. The analogue control is achieved by means of a varicap $C_{V,\text{tot}}(v_{\text{tune,eff}})$ allowing continuous frequency tuning over a certain range. The digital control is realised as a bank of switchable capacitances $C_{\text{tank}}(\text{band})$, which allows to shift the centre frequency $f_{VCO,0}$ of the range covered with the analogue control in a discrete way. The final oscillation frequency is thus given by:

$$f_{VCO} = \frac{1}{2\pi \cdot \sqrt{L_{\text{TANK}} \cdot (C_{\text{tank}}(\text{band}) + C_{V,\text{tot}}(v_{\text{tune,eff}}))}} \quad (4.6)$$

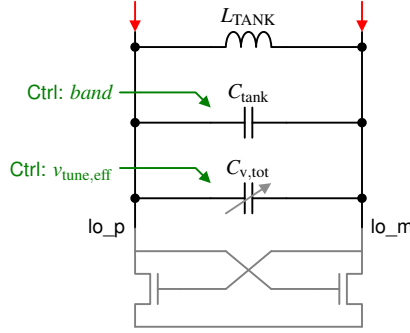


Figure 4.3.: Resonant elements of the VCO.

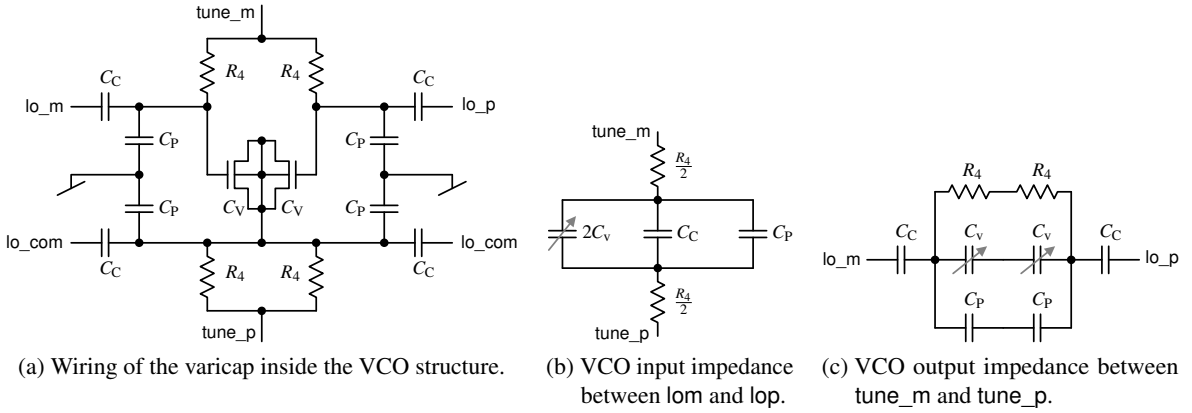


Figure 4.4.: Wiring of the varicap inside the VCO and resulting input/output impedances.

with L_{TANK} the oscillator inductance, $C_{\text{tank}}(\text{band})$ the bank capacitance for the selected *band*, $C_{v,\text{tot}}(v_{\text{tune,eff}})$ the variable capacitance for the effective tuning voltage $v_{\text{tune,eff}}$.

The previous equations show that the primary parameter needed for a correct VCO model is K_{VCO} . It is present in the time- and frequency-domain equations (by means of the variable capacitance) affecting not only the VCO behaviour but also the PLL stability (Eq. (4.3)). The value of K_{VCO} is directly related to the characteristic of the varicap, which is the circuit element used to perform the frequency tuning. Therefore, a detailed study of the varicap and its usage in the VCO circuit has to be done. Figure 4.4a depicts the varicap configuration inside the VCO structure with biasing resistances R_4 , coupling capacitances C_C , and parasitic capacitances C_P . The equivalent capacitances seen from the filter (C_4 , Figure 4.4b) and from the oscillator terminals ($C_{v,\text{tot}}$, Figure 4.4c) are:

$$C_4 = 2C_v + C_C + C_P \quad (4.7)$$

$$C_{v,\text{tot}} = \left(\frac{2}{C_C} + \frac{2}{C_v + C_P} \right)^{-1} \quad \text{with} \quad \frac{C_P}{C_C} \cong 1 \% \quad (\text{technology dependent}) \quad (4.8)$$

The nonlinear dependency of the varicap density D_{cv} from the effective tuning voltage $v_{\text{tune,eff}}$ at its terminals can be accurately described using a hyperbolic tangent function:

$$D_{\text{cv}}(v_{\text{tune,eff}}) = D_{\text{CV,AMP}} \cdot \tanh \left(\frac{\frac{dD_{\text{CV},0}}{dv_{\text{tune,eff}}}}{D_{\text{CV,AMP}}} \cdot (v_{\text{tune,eff}} - V_0) \right) + D_{\text{CV},0} \quad (4.9)$$

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

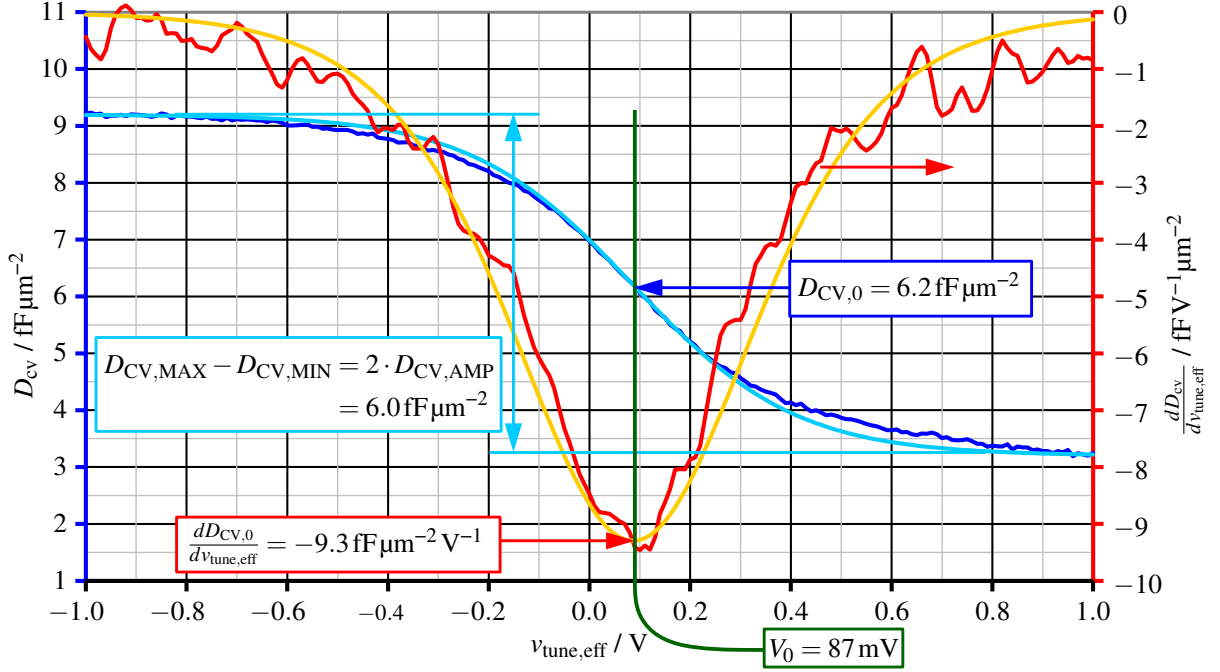


Figure 4.5.: Fitting of the varicap density and its derivative to the TSMC 0.18μm process using a hyperbolic tangent function (Eqs. (4.9) and (4.10)).

with the following technology-dependent parameters (Figure 4.5):

V_0 : Offset tuning voltage for the point with the steepest slope $\frac{dD_{cv}}{dv_{tune,eff}}$ of the measured varicap density.

$D_{CV,0} = D_{cv}(V_0)$: Varicap density measured for V_0 .

$D_{CV,AMP}$: Varicap density modulation amplitude.

$\frac{dD_{CV,0}}{dv_{tune,eff}} = \left. \frac{dD_{cv}}{dv_{tune,eff}} \right|_{V_0}$: Slope of D_{cv} measured at V_0 .

By deriving the varicap density function with respect to the tuning voltage $v_{tune,eff}$, we obtain:

$$\frac{dD_{cv}}{dv_{tune,eff}} = \frac{dD_{CV,0}}{dv_{tune,eff}} \cdot \left(1 - \left(\frac{D_{cv}(v_{tune,eff}) - D_{CV,0}}{D_{CV,AMP}} \right)^2 \right) \quad (4.10)$$

Then, by defining the area A_{CV} of the varicap, it is possible to calculate its absolute value and derivative:

$$C_v = A_{CV} \cdot D_{cv} \quad \frac{dC_v}{dv_{tune,eff}} = A_{CV} \cdot \frac{dD_{cv}}{dv_{tune,eff}} \quad (4.11)$$

To calculate K_{VCO} , we also need the derivative $\frac{dC_{v,tot}}{dv_{tune,eff}}$ of the total equivalent varicap $C_{v,tot}$ seen by the VCO between the terminals lo_p and lo_m taking into account the coupling and parasitic capacitances present in the circuit (Figure 4.4a):

$$\frac{dC_{v,tot}}{dv_{tune,eff}} = \frac{1}{2} \cdot \frac{1}{\left(1 + \frac{C_v}{C_C} + \frac{C_P}{C_C} \right)^2} \cdot \frac{dC_v}{dv_{tune,eff}} \quad (4.12)$$

For a given area of the coupling capacitors and varicap, the optimum point can be calculated by searching the maximum of Eq. (4.12). In the case of the TSMC 0.18 μm process, the density of the coupling capacitance D_{CC} is 0.8 fF/ μm^2 and the density of the varicap $D_{CV,0} = D_{cv}(87 \text{ mV}) = 6.2 \text{ fF}/\mu\text{m}^2$ in Figure 4.5. The optimum point appears for a ratio $\frac{C_C}{C_{V,0}}$ equal to:

$$\frac{C_C}{C_{V,0}} = \frac{1}{2 \cdot (1 + \frac{C_P}{C_C})} \cdot \left(1 + \sqrt{1 + 8 \cdot \frac{D_{CC}}{D_{CV,0}} \cdot (1 + \frac{C_P}{C_C})} \right) \quad (4.13)$$

The resulting $\frac{C_C}{C_{V,0}} \approx 1.2$ constitutes a lower limit for the choice of the ratio between the coupling capacitance and varicap. However, in that case the $k_{vco}(f_{vco})$ curve (Figure 4.7d on page 57) won't be very symmetric around the VCO's centre frequency. Therefore, $\frac{C_C}{C_{V,0}}$ needs to be increased reasonably to keep the parasitic capacitance C_P (which is coupled through a fixed ratio to C_C in Eq. (4.8)) as low as possible with respect to the varicap (Figure 4.4c). From experience, a $\frac{C_C}{C_{V,0}} = 4$ constitutes a good trade-off and has been used for the VCO in the design examples presented hereafter.

Using Eqs. (4.6) and (4.12), we can finally calculate K_{vco} :

$$K_{vco} = \frac{df_{vco}}{dv_{tune,eff}} = 2\pi^2 \cdot f_{vco}^3 \cdot L_{TANK} \cdot \left(-\frac{dC_{v,tot}}{dv_{tune,eff}} \right) \quad (4.14)$$

Using the technology constants introduced for the varicap density given by Eq. (4.9) and choosing the ratios $\frac{C_C}{C_{V,0}}$ and $\frac{C_P}{C_C}$ for the capacitances, the varicap area A_{CV} and its related capacitances C_C and C_P can be calculated to optimise the VCO to have a maximal K_{vco} at a certain target frequency $f_{VCO,OPT}$. This optimal frequency is supposed to be at the inflection point of $D_{cv}(v_{tune,eff})$ at V_0 . This maximal K_{vco} is used for the PLL stability calculation with the help of Eqs. (4.3) and (4.4) described in Section 4.3.1. For accurate simulation, the instantaneous value of K_{vco} is always used.

4.3.3. Design and Implementation of the Component Models

During the design and implementation phase, the functional specification of each component developed together with the RF designer is translated into an executable model. Thanks to the expressiveness of VHDL-AMS, which is close to the algorithmic and mathematical notation used to describe digital and analogue behaviour, it is rather straightforward to model the specified behaviour. Common to all models is the systematic usage of assertions to check for consistent parametrisation of the model and compliance of the model state with the modelling assumptions. To facilitate code comprehension and maintenance, a consistent coding style is enforced through peer-review establishing/encouraging for instance:

- A naming convention,
- Consistent commenting of each declared constant/quantity/signal regarding its purpose and physical unit,
- The usage of common coding patterns for digital behaviour,
- The implementation of each model effect as local and independent from each other as possible.

To facilitate the parallel development/usage/maintenance of the models, their sources, related documentation, and their simulation environment are managed by a revision control system, e.g., Subversion in the case of the RF_TRX library.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

Listing 4.1: Overall structure of the VHDL-AMS models developed for the RF_TRX library.

```
1  entity <model>
2    generic (
3      -- Primary parameters of each abstraction level
4    );
5    port (
6      -- Pin-accurate port interface
7    );
8  begin
9    -- Interface level assertions
10 end;
11
12 architecture <abstraction_level>
13   -- Calculation of secondary parameters
14   -- Signal, terminal, and quantity declarations
15 begin
16   -- Debug code and architecture level assertions
17   -- Behavioural description
18   -- Compute design data (e.g., performance estimates)
19 end;
```

The VHDL-AMS models developed in this way for the RF_TRX library have all the same overall structure, which is outlined in Listing 4.1. There is one common **entity** definition for the interface of the component and one or more **architectures** implementing the behavioural or structural description of the model on different abstraction levels or for different design tasks (e.g., for *top-down design* and *bottom-up verification*). The entity defines the pin-accurate port interface of the component model so that structural model descriptions can be mostly reused throughout the whole design process. To this end, structural models usually do not instantiate directly an entity with an architecture, but use an identical **component** declaration from a package. A **configuration** can then specify during elaboration which entity/architecture pair shall be used for the simulation at hand. Moreover, multi-level simulations with transistor level models replacing behavioural model of certain components during bottom-up validation are facilitated by this approach (Sections 4.3.4 and 4.4.3). If during the design process the interface changes, the developed models can be usually easily adapted by adding a thin wrapper instantiating the model and mapping the generics and ports from the new to the old interface.

The entity definition contains the **generic** parameters for all architectures. Each architecture documents which subset of these primary parameters it uses for the calculation of its internal secondary parameters. Only after the calculation of the secondary parameters, the architecture declares its internal signals, terminals, and quantities. The architecture body contains first debug code and architecture level assertions to check the calculated secondary parameters against fundamental model assumptions and to monitor the state of the model. Assumptions that can be already checked on the interface level, e.g., the consistency of the supplied generic parameters or the value range for port signals, are implemented in form of assertions in the body of the entity. Each architecture implementation can rely on these guarantees and must, as a consequence, fulfil its part of the contract concerning the allowed value ranges of the output signals. Then follows the behavioural implementation itself. In the last part of the architecture body, some design data, e.g., performance estimates, are calculated, which do not contribute to the component's behaviour but give the designer insight into its state, functioning, and performance.

In the RF_TRX library, the PFD, CP, dividers, and Σ - Δ modulator models have a single architecture, which satisfies the needs for the top-down design and bottom-up verification phases. Their top-down design parameters are also used during the bottom-up phase, which provides additional details like delays or mismatches. The *Phase/Frequency Detector (PFD) model* is implemented as a digital behaviour description (two D-flip-flops and one AND-gate) augmented by generics to specify the propagation delays of the gates, which are back-annotated during the bottom-up phase. The *differential charge pump model* is implemented as ideally switched current sources with internal resistance. Their values are determined as a multiple of a measured input bias current, as they are implemented as current mirrors on transistor level. Mismatches can be specified for the bottom-up verification. This level of precision is sufficient. For a more detailed study of the charge pump's impact on the PLL performance (especially due to the switching effects from the transistors), its SPICE component model can be integrated into the system level VHDL-AMS model and then simulated (Section 4.4.3). Even though it is not standardised, this approach is nowadays supported by many simulators. As a general strategy for the implementation of the RF_TRX library, component models consider only those non-ideal effects, which have a direct impact on the system design and can be approximately quantified/described without retaining all the details of the transistor level implementation. Effects, which cannot be easily quantified or which behaviour cannot be abstracted away from the transistor level implementation, are left unconsidered in the behavioural models. Their impact is only evaluated after the transistor level implementation during the bottom-up verification phase using selectively the transistor level models in the behavioural system model.

The implementation of the *differential loop filter model* is a bit more complex. The stability analysis of the PLL (Section 4.3.1) shows that the model needs to describe the electrical RC network. That is why it has been implemented using branch equations instead of a SPICE-like structural description. This allows to use simultaneous **if**-statements to implement the filter orders 1, 2, and 3 in the same model (Listing 4.2). The capacitors' behaviour is described in its integral form (using 'integ') instead of its differential form ('dot'), as this showed better numerical behaviour in simulation. For the top-down design phase, the RF designer can specify in the generics, besides the filter cut-off frequency $f_{\text{CUT-OFF}}$ and its order, the phase margin Φ_{MARGIN} to be realised for the PLL characterised by the charge pump sensibility K_{PD} , the VCO frequency sensibility K_{VCO} , and the frequency division factor N_{DIV} . This information is used in the *top-down architecture* to calculate the R and C values of the filter taking into account the device ratios $\frac{R_3}{R_2}$ and $\frac{C_3}{C_1}$ as well as the (possibly back-annotated) VCO input impedance defined by R_4 and C_4 . The *bottom-up architecture* allows to directly specify the R and C values.

The *VCO model* constitutes to date the most complex model of the RF_TRX library. Five different interfaces (i.e., entities) model the RF output signal at various levels of abstraction (Figure 4.6):

Instantaneous frequency: is directly calculated using Eq. (4.6) and thus suppresses the RF effects in simulation. As a consequence, the integration step size of the analogue solver can be increased up to one order of magnitude below the time constant of the PLL, which considerably accelerates transient simulation. In combination with other base-band modelling approaches [36], this abstraction allows to use the VCO output for up- and down-conversion in the frequency domain.

Single-ended and differential digital signals: are completely generated in the Discrete Event (DE) domain by sampling f_{VCO} at the state change of the digital signal lo and calculating from it the delay till the next state change:

$$lo \leftarrow \overline{lo} \text{ after } \frac{1}{|f_{\text{VCO}}(t)|} \quad f_{\text{VCO}} \gg f_{\text{REF,PLL}} \quad (4.15)$$

Listing 4.2: Architecture implementing the differential loop filter behaviour of filter orders 1, 2, and 3.

```

1  architecture td_detailed of loop_filter_diff is
2    -- Initialise the C1, R2, C2, R3, and C3 constants from the generics...
3
4    -- Declaration of the voltage [V] and current [A] branch quantities.
5    quantity v_c1 across i_c1 through tune_p_in to tune_m_in;
6    quantity v_2 across i_2 through tune_p_in to tune_m_in;
7    quantity v_r3_1 across i_r3_1 through tune_p_in to tune_p_out;
8    quantity v_r3_2 across i_r3_2 through tune_m_out to tune_m_in;
9    quantity v_c3 across i_c3 through tune_p_out to tune_m_out;
10 begin -- architecture td_detailed
11    -- Debug code...
12
13    -- Behaviour implementation:
14    v_c1 == (1.0 / C1) * i_c1'integ;
15    if order >= 2 use
16        v_2 == R2 * i_2 + (1.0 / C2) * i_2'integ;
17    else
18        i_2 == 0.0;
19    end use;
20    if order >= 3 use
21        v_r3_1 == 0.5 * R3 * i_r3_1;
22        v_r3_2 == 0.5 * R3 * i_r3_2;
23        v_c3 == (1.0 / C3) * i_c3'integ;
24    else
25        v_r3_1 == 0.0;
26        v_r3_2 == 0.0;
27        i_c3 == 0.0;
28    end use;
29 end architecture td_detailed;

```

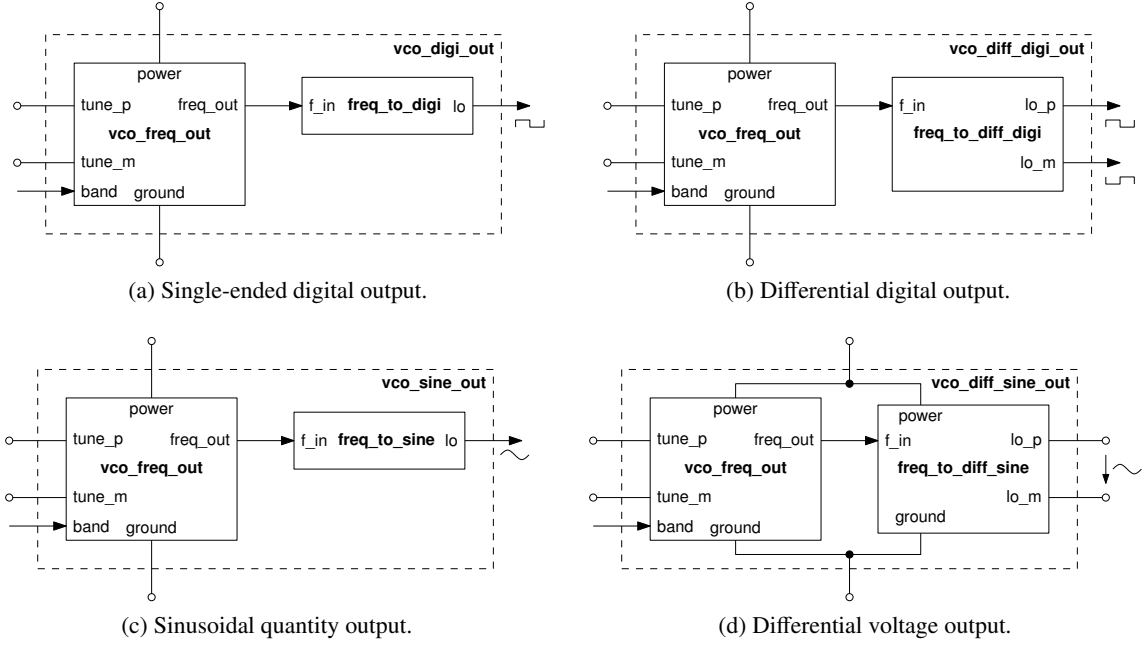


Figure 4.6.: VCO models with transient outputs.

This decoupling of the analogue and discrete solvers allows the analogue solver to increase its integration step size. This abstraction is useful if the VCO output is fed directly into digital blocks.

Sinusoidal quantity or differential voltage signals: are generated from the specified amplitude and the current value of the phase, which is calculated by integrating the instantaneous f_{vco} over time:

$$\varphi_{vco}(t) = 2\pi \int_{0_s}^t f_{vco}(t') dt' + \varphi_0 \quad (4.16)$$

$$v_{lo}(t) = V_{LO,AMP} \cdot \sin \varphi_{vco}(t) \quad (4.17)$$

This generation of the RF signal in the Continuous-Time (CT) domain considerably slows down the analogue solver, which integration step size needs to stay one order of magnitude below f_{vco} . Nevertheless, it is more efficient than using a full LC oscillator model. This detailed abstraction may be necessary to interface analogue blocks operating at RF and to do spectral analysis.

All five interfaces use the instantaneous frequency f_{vco} (Eq. (4.6)). That is why it was decided to reuse the VCO model with frequency output inside the models with digital or sinusoidal output (Figure 4.6). The *frequency to digital* and *frequency to sinusoidal conversions* have been factorised into separate models, which can be reused independently. To this end, the frequency to digital converters provide two architectures. One implements the previously described *discrete* conversion. The other implements an *ideal* conversion based on the integration of the input frequency like the frequency to sinusoidal converter does (Eqs. (4.16) and (4.17)). This is necessary to avoid phase errors in cases, for which the input signal cannot be considered as quasi-steady compared to the output signal.

The VCO model with instantaneous frequency output is thus the single place for implementing the functional specification from Section 4.3.1 on three abstraction levels corresponding to three architectures.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

Listing 4.3: Entity declaration of the VCO with frequency output model.

```

1  entity vco_freq_out is
2    generic ( -- Primary parameters of each abstraction level
3      F_OPT      : real      := 830.0e6;    -- Centre frequency to be optimised [Hz]
4      K_OPT      : real      := 60.0e6;    -- Frequency sensibility [Hz/V]
5      F_MIN      : real      := 750.0e6;    -- Minimum centre frequency [Hz]
6      F_MAX      : real      := 890.0e6;    -- Maximum centre frequency [Hz]
7      N_BITS     : positive := 3;          -- Number of bits for band selection [1]
8      N_BANDS    : positive := 8;          -- Number of bands
9      RATIO_CC_CV0 : real     := 4.0;        -- Ratio of C_C and C_VO [1]
10     RATIO_CP_CC  : real     := 0.01;       -- Ratio of C_P and C_C [1]
11     L_TANK       : real     := 5.0e-9;     -- Tank inductance [H]
12     R_IN         : real     := 20.0e3;     -- Input resistance [Ohm]
13     -- Technology constants of the varicap
14     V_0          : real := 0.087;          -- Offset voltage [V]
15     D_CV0        : real := 6.2e-3;        -- Varicap cap. density at V_0 [F/m**2]
16     DDCV0_DV     : real := -9.3e-3;       -- Slope of D_CV at V_0 [F/(m**2 * V)]
17     D_CV_AMP     : real := 3.0e-3;        -- Cap. density modulation amp. [F/m**2]
18     -- Component parameters from real design for bottom-up verification
19     S_CV_DESIGN   : real := 3.63e-10;     -- Varicap surface [m**2]
20     C_TANK_MIN_DESIGN : real := 5.47e-12; -- Minimum tank capacitance [F]
21     C_TANK_MAX_DESIGN : real := 8.08e-12; -- Maximum tank capacitance [F]
22     -- Power supply voltage range
23     V_POWER_MIN   : real := 0.9;          -- Minimum supply voltage [V]
24     V_POWER_MAX   : real := 1.8;          -- Maximum supply voltage [V]
25     DEBUG         : boolean := false);    -- Debug instance
26  port ( -- Pin-accurate
27    signal band : in std_logic_vector(N_BITS-1 downto 0)
28              := std_logic_vector(to_unsigned(0, N_BITS)); -- Band selector
29    terminal power, ground : electrical;    -- Power supply terminals
30    terminal tune_p, tune_m : electrical;    -- Tuning terminals
31    quantity f_out         : out real);    -- VCO frequency [Hz]
32  begin
33    -- Interface level assertions...
34  end entity vco_freq_out;

```

Its entity definition is given in Listing 4.3 showing all available generics and ports. Table 4.1 gives an overview on the three architectures, their purpose, implemented behaviours, used generic parameters, derived internal constants, and calculated transient values. The *ideal* architecture implements the band switching capability and uses a constant frequency sensitivity K_{VCO} . No input impedance is considered at the tuning terminals. Two *detailed* architectures (one for top-down design, the other for bottom-up verification) implement the full nonlinear behaviour of the VCO, as specified in Section 4.3.2, including band switching, nonlinear varicap, nonlinear input impedance depending on $v_{tune,eff}$, and calculation of the instantaneous K_{VCO} . The *top-down variant* uses a subset of the generics defined in the entity, which represent the component specification (e.g., number of bands, frequency range, $K_{VCO,OPT}$ at $f_{VCO,OPT}$) and the target technology (fitting parameters of Eq. (4.9)) to calculate the internal (secondary) device parameters (e.g., A_{CV} , C_C , C_P , and the parameters of the tank capacitance bank). The *bottom-up variant* directly uses the device parameters passed through the generics. Listing 4.4 shows how the nonlinear behaviour of the VCO is implemented in the *td_detailed* architecture. The behavioural description in the *bu_detailed* architecture does not differ, only the secondary parameters are initialised differently.

Table 4.1.: Overview on the architectures of the VCO model with band switching capability and instantaneous frequency output ($v_{co_freq_out}$).

Architecture	ideal	td_detailed	bu_detailed
Use case	<ul style="list-style-type: none"> To assess early system specifications. 	<ul style="list-style-type: none"> To assess nonlinear transient behaviour and derive VCO implementation specifications. 	<ul style="list-style-type: none"> To assess impact of VCO implementation on the system level.
Model features	<ul style="list-style-type: none"> Band switching Constant $k_{vco} = k_{opt}$ No input load Checks supply voltage, but draws no power 	<ul style="list-style-type: none"> Band switching Nonlinear k_{vco} Nonlinear input load Checks supply voltage, but draws no power 	<ul style="list-style-type: none"> Band switching Nonlinear k_{vco} Nonlinear input load Checks supply voltage, but draws no power
Used generics	<ul style="list-style-type: none"> Constant K_{vco} (K_{OPT}) Centre frequencies range f_{MIN} (F_{MIN}) to f_{MAX} (F_{MAX}) Number of bands (N_BANDS, N_BITS) Supply voltage range (V_POWER_MIN, V_POWER_MAX) 	<ul style="list-style-type: none"> Target K_{vco} (K_{OPT}) at centre f_{vco} (F_{OPT}) Centre frequencies range f_{MIN} (F_{MIN}) to f_{MAX} (F_{MAX}) Number of bands (N_BANDS, N_BITS) Ratios $C_C/C_{V,0}$ ($RATIO_CC_CV0$) and C_P/C_C ($RATIO_CP_CC$) External tank inductance L_{TANK} (L_TANK) Input resistance R_4 (R_IN) Varicap technology parameters V_0 (V_0), $\frac{dC_{V,0}}{dV_{tune,eff}}$ ($DDCV0_DV$), $D_{CV,AMP}$ (D_CV_AMP) Supply voltage range (V_POWER_MIN, V_POWER_MAX) 	<ul style="list-style-type: none"> Number of bands (N_BANDS, N_BITS) Ratios $C_C/C_{V,0}$ ($RATIO_CC_CV0$) and C_P/C_C ($RATIO_CP_CC$) External tank inductance L_{TANK} (L_TANK) Input resistance R_4 (R_IN) Varicap technology parameters V_0 (V_0), $\frac{dC_{V,0}}{dV_{tune,eff}}$ ($DDCV0_DV$), $D_{CV,AMP}$ (D_CV_AMP) Varicap area A_{CV} (S_CV_DESIGN) Tank capacitance range C_{TANK} ($band$) ($C_TANK_MIN_DESIGN$, $C_TANK_MAX_DESIGN$) Supply voltage range (V_POWER_MIN, V_POWER_MAX)
Derived constants	<ul style="list-style-type: none"> Centre frequencies distance (F_STEP) 	<ul style="list-style-type: none"> Varicap parameters A_{CV} (S_CV), $C_V(V_0)$ (C_V0), $\frac{dC_V}{dV_{tune,eff}} _{V_0}$ ($DDCV0_DV$) Coupling capacitance C_C (C_C) Parasitic capacitance C_P (C_P) Input capacitance $C_4(V_0)$ (C_IN0) Total varicap capacitance $C_{V,tot}(V_0)$ (C_V0TOT), $\frac{dC_{V,tot}}{dV_{tune,eff}} _{V_0}$ ($DDCV0TOT_DV$) Tank capacitance range (C_TANK_DESIGN, C_TANK_MAX, C_TANK_STEP) 	<ul style="list-style-type: none"> Varicap parameters $C_V(V_0)$ (C_V0), $\frac{dC_V}{dV_{tune,eff}} _{V_0}$ ($DDCV0_DV$) Coupling capacitance C_C (C_C) Parasitic capacitance C_P (C_P) Input capacitance $C_4(V_0)$ (C_IN0) Total varicap capacitance $C_{V,tot}(V_0)$ (C_V0TOT), $\frac{dC_{V,tot}}{dV_{tune,eff}} _{V_0}$ ($DDCV0TOT_DV$) Tank capacitance step size C_TANK_STEP Centre frequencies range (F_MIN_DESIGN, F_MAX_DESIGN, F_STEP_DESIGN)
Calculated transients	<ul style="list-style-type: none"> Instantaneous $f_{vco} = f_{VCO,0}(band) + K_{VCO} \cdot V_{tune}$ 	<ul style="list-style-type: none"> Instantaneous K_{vco}, f_{vco} 	<ul style="list-style-type: none"> Instantaneous K_{vco}, f_{vco}

4.3. Modelling the Frequency Synthesiser

Listing 4.4: Top-down detailed architecture of the VCO with frequency output model.

```

1  architecture td_detailed of vco_freq_out is
2      -- Definition of support functions...
3      -- Calculation of secondary parameters...
4      -- Quantity and Signals declaration...
5  begin
6      -- Debug code and architecture level assertions...
7
8      -- Behaviour implementation:
9      -- Power supply
10     i_power == 0.0;
11     -- load at input
12     c_in == 2.0 * c_v + C_P + C_C;
13     v_tune_eff == (1.0 / c_in) * i_tune'integ;
14     v_tune == R_IN * i_tune + v_tune_eff;
15     -- Band switching
16     c_tank <= C_TANK_MAX - real(to_integer(unsigned(band))) * C_TANK_STEP;
17     break on c_tank;
18     -- Varicap capacitance
19     d_cv == D_CV_AMP * tanh((DDCV0_DV / D_CV_AMP) * (v_tune_eff - V_0)) + D_CV0;
20     c_v == S_CV * d_cv;
21     dcv_dvtune_eff == S_CV * (1.0 - ((d_cv - D_CV0) / D_CV_AMP)**2) * DDCV0_DV;
22     c_vtot == 1.0 / (2.0 / C_C + 2.0 / (c_v + C_P));
23     dcvtot_dvtune_eff == 0.5 * (C_C / (c_v + C_P + C_C))**2 * dcv_dvtune_eff;
24     -- Output frequency
25     f_out == 1.0 / (MATH_2_PI * sqrt(L_TANK * (c_tank + c_vtot)));
26
27     -- Compute design data (e.g. performance estimates):
28     -- Voltage sensitivity of VCO frequency
29     k_vco == 2.0 * MATH_PI**2 * f_out**3 * L_TANK * (-dcvtot_dvtune_eff);
30 end architecture td_detailed;

```

Several programmable divider models have been implemented [15] with a focus on a flexible interface (size of control word, digital input or instantaneous frequency input) and parameterisable digital behaviour (e.g., minimal/maximal division ratio, control word treated as absolute division ratio or offset to minimal division ratio). For a two-stage divider architecture, a divider model with integrated control logic for a prescaler has been implemented.

For fractional division ratio control and noise shaping, a Σ - Δ modulator model has been developed [15]. It describes a MASH structure [152] of 1st to 3rd order on RTL using **generate** statements. The number of bits representing the integer and fractional part can be adjusted to change the modulator resolution during architecture exploration in order to meet the frequency synthesiser specification and be able to reuse it in other projects.

The divider and Σ - Δ modulator models are primarily targeted to be used during the top-down design phase. During bottom-up verification, they can be replaced by the gate level HDL models with detailed annotated delays exported by the synthesis and place & route tools.

To summarise, Table 4.2 gives an overview on the models currently available in the RF_TRX library. For some components, several models are available offering different interfaces. Depending on the model, several architectures may be available to adapt the model to the design task at hand. The power consumption of these components is not yet explicitly modelled as it was not identified to be a relevant use case for the current state of development of the RF_TRX library. However, the power supply terminals are included in each model, which test that the applied supply voltage has the correct polarity and remains in the specified range. This is important to facilitate the bottom-up verification of the connectivity.

4.3.4. Validation of the Frequency Synthesiser Component Models

Each component model has a dedicated test bench instantiating the model through a **component** declaration and providing configurations to bind each architecture to the component. Thus, all architectures can be validated with the same test bench. When realisable with reasonable effort, the test benches implement automated measurements and checks to validate the model behaviour. This can be done, e.g., in form of processes, which calculate performance figures or monitor output signals by comparing them to some kind of reference signals or expected envelope. Assertions can be used to check for fundamental model assumptions. Simulation control scripts and waveform display setups for each configuration ensure the reproducibility of simulation runs. Compilation and simulation of all models and test benches is handled through a common *build system* implemented for Mentor Graphics ADVance MS™ (ADMS) effectively putting an automated unit testing system in place. This drastically reduces the modification/simulation turn-around times easing the evaluation of the impact of code modifications. This lowers the risk that revisions committed to the Subversion repository contain code modifications that break the model or on it depending models.

As an example for such an elaborate component test bench setup, the validation of the band switching VCO model with differential digital output is discussed in the following. The VCO is tested by instantiating it inside the test bench and connecting it to the power supply. The VCO internally checks that the power supply has the correct polarity and that it is in the specified range. The test bench applies a saw tooth voltage v_{tune} to the VCO's tune input. Each saw tooth period, the band is switched by incrementing the control word *band*. Thus all frequencies that can be generated by the VCO are successively scanned through. Figure 4.7 shows the simulation results of the VCO test bench after an automated transient analysis for 80 μ s and loading the simulation results with the supplied waveform display setup in the waveform viewer. The VCO used in this simulation has 8 bands to cover a frequency range from 750 MHz to 890 MHz. Its parametrisation corresponds to the default values for the entity's

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

Table 4.2.: Overview on the RF_TRX model library.

Component	Entity	Architectures	Type	Description
PFD	pdf_digi_diff	rtl	RTL	Digital PFD considering delays of flip-flops and reset.
Charge pump	charge_pump_diff	detailed	AMS	Commuting current sources with switch resistances, mismatches, and slew rate.
Loop filter	loop_filter_diff	td_detailed, bu_detailed	A	RC network differential filter of order 1, 2, or 3.
VCO	vco_freq_out	ideal, td_detailed, bu_detailed	A	Band switching VCO with instantaneous frequency output and nonlinear input impedance, $f_{\text{vco}}(C_v(v_{\text{tune,eff}}))$ for <i>detailed</i> architectures.
	vco_digi_out, vco_diff_digi_out	ideal, ideal_discrete, td_detailed, td_detailed_discrete, bu_detailed, bu_detailed_discrete	AMS	Band switching VCO with digital output and nonlinear input impedance, $f_{\text{vco}}(C_v(v_{\text{tune,eff}}))$ for <i>detailed</i> architectures.
	vco_sine_out, vco_diff_sine_out	ideal, td_detailed, bu_detailed	A	Band switching VCO with sinusoidal output and nonlinear input impedance, $f_{\text{vco}}(C_v(v_{\text{tune,eff}}))$ for <i>detailed</i> architectures.
$f \rightarrow t$ converters	freq_to_digi, freq_to_diff_digi	ideal, discrete	AMS	Instantaneous frequency to digital signal.
	freq_to_sine, freq_to_diff_sine	ideal	A	Instantaneous frequency to sine wave.
Dividers	divider_prog_digi	ideal	D	Programmable digital divider.
	divider_prog_freq_digi	ideal	AMS	Programmable digital divider with instantaneous frequency input.
	divider_prog_presc_ctrl	dual_modulus	D	Programmable divider with control logic for dual-modulus prescaler.
$\Sigma\Delta$ modulator	sigma_delta_mash	rtl	RTL	MASH of order 1, 2, or 3.
Pulse shaper	bit_stream_modulator	discrete, ramped, raised_cosine	D	Shaping input bit stream to have <i>discrete</i> , <i>ramped</i> , or <i>raised cosine</i> pulse shape.

A: Analogue model.

AMS: Mixed-signal model.

D: Digital model.

RTL: Synthesisable digital model.

ideal: Ideal (usually linear) behaviour.

discrete: Discrete behaviour based on sampling of continuous-time signals.

td: Detailed behaviour (e.g., considering delays and important nonlinear effects).

bu: “Bottom-up” model for architecture exploration.

bu: “Bottom-up” model for verification of transistor level implementation.

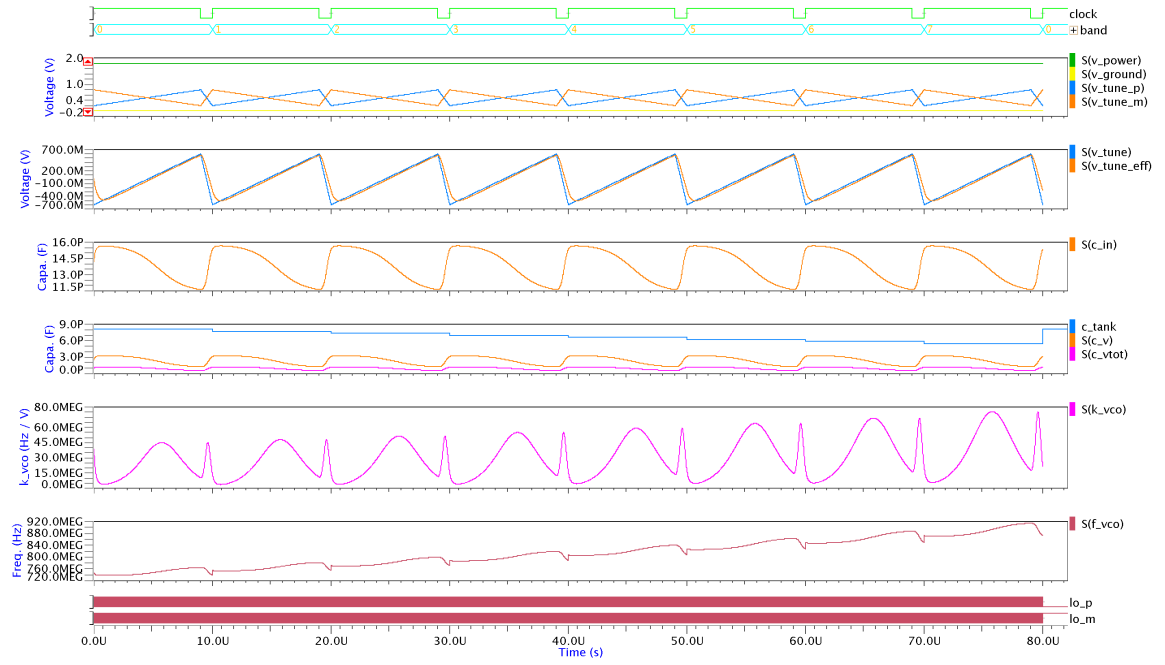
Table 4.3.: Comparison of the simulation performances between the different architecture and output options for the VCO model. Each VCO model configuration underwent a transient analysis ($t_{\text{stop}} = 80 \mu\text{s}$, $\text{eps} = 10^{-6}$, cf. Figure 4.7) with ADMS 2008.2 on an Intel Core 2 Quad 2.66 GHz CPU, 4 MB cache, 4 GB RAM running Linux 2.6.9 x86_64. For each configuration, the chosen maximum integration step size h_{max} and the kind of frequency to transient signal conversion method are given.

	Architecture	ideal	td_detailed	bu_detailed
vco_freq_out	CPU time	290 ms	630 ms	630 ms
($h_{\text{max}} = 10 \text{ ns}$)	Steps	8025	8029	8029
vco_diff_digi_out	CPU time	20 780 ms	26 250 ms	26 180 ms
($h_{\text{max}} = 200 \text{ ps}$, ideal $f \rightarrow d$)	Steps	483 495	488 828	489 202
vco_diff_digi_out	CPU time	620 ms	1060 ms	1020 ms
($h_{\text{max}} = 10 \text{ ns}$, discrete $f \rightarrow d$)	Steps	8050	8048	8048
vco_diff_sine_out	CPU time	52 930 ms	81 780 ms	81 360 ms
($h_{\text{max}} = 200 \text{ ps}$, ideal $f \rightarrow \sin$)	Steps	1 193 872	1 178 317	1 178 086

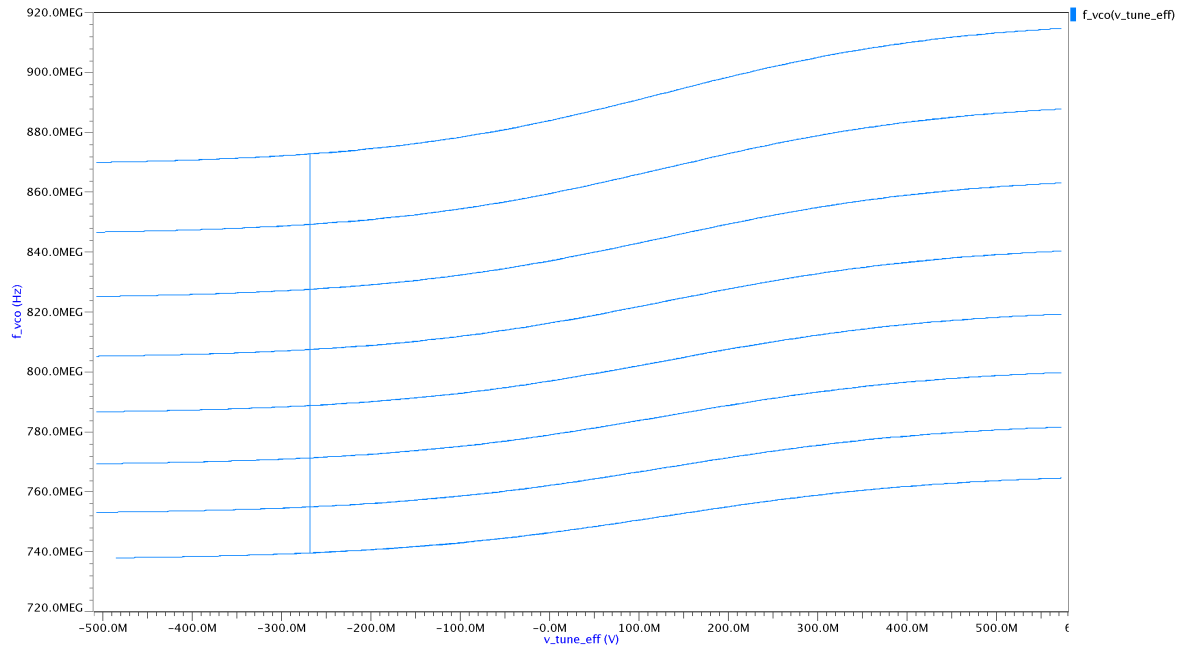
generic parameters already shown in Listing 4.3. Besides the transient signals (Figure 4.7a), additional graphs visualise the nonlinear dependency of the VCO's output frequency $f_{\text{vco}}(v_{\text{tune,eff}})$ and frequency sensitivity $k_{\text{vco}}(v_{\text{tune,eff}})$ from the tuning voltage $v_{\text{tune,eff}}$ stemming from the varicap. For the design of a VCO, the nonlinear relation of $k_{\text{vco}}(f_{\text{vco}})$ is especially important and thus made available by the waveform display setup (Figure 4.7d). It allows the designer to ensure that the output frequency range of the VCO is able to cover the frequency bands imposed by the RF application without gaps as well as with a sufficiently high and symmetrical $k_{\text{vco}}(f_{\text{vco}})$ curve for a stable frequency synthesis.

The simulation of the VCO models with different architectures and outputs in the described test bench is very fast, as shown by the simulation performance measurements summarised in Table 4.3. The parameterisation of the test bench is the same as the one used for the simulation results already presented in Figure 4.7. Especially, the VCO models with instantaneous frequency output and discrete frequency to digital output signal conversion simulate in only about a second. Even the VCO models with ideal frequency to digital and sinusoidal signal conversion simulate within less than 0.5 min and 1.5 min, respectively. The results also show the expected simulation speed advantage for the *ideal* architecture variants with linear behaviour. However, the *td_detailed* and *bu_detailed* architecture variants, which include all major nonlinear effects, do only extend the required CPU time by a factor of 2.17 in the worst case (*vco_freq_out*) and 1.55 for the best case (*vco_diff_sine_out*). In the latter case the frequency to sinusoidal conversion becomes the dominant factor limiting the simulation speed. The number of solution steps is more or less independent from the architecture choice, as it is governed by the choice of the integration step size h_{max} , which has been chosen to correctly sample all analogue signals. Thus, the presented test bench allows not only the validation of the behavioural VCO models, but gives the RF designer an efficient tool at hand to design a VCO for a given specification. He can directly supply the specification as parameters to the top-down architecture (*td_detailed*) of the VCO model (preferably *vco_freq_out* or *vco_diff_digi_out*) as described in Section 4.3.3. The model internally calculates suitable parameters for the varicap as well as the coupling and tank capacitors, which the designer can use as a starting point for the transistor level implementation.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems



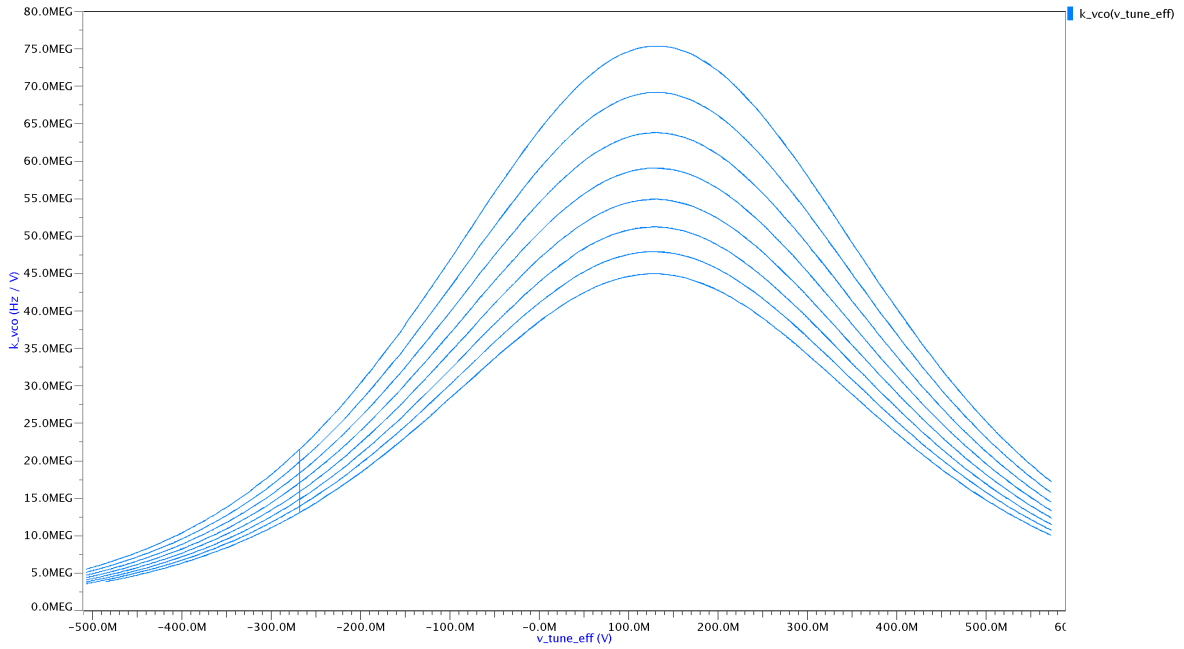
(a) Transient signals.



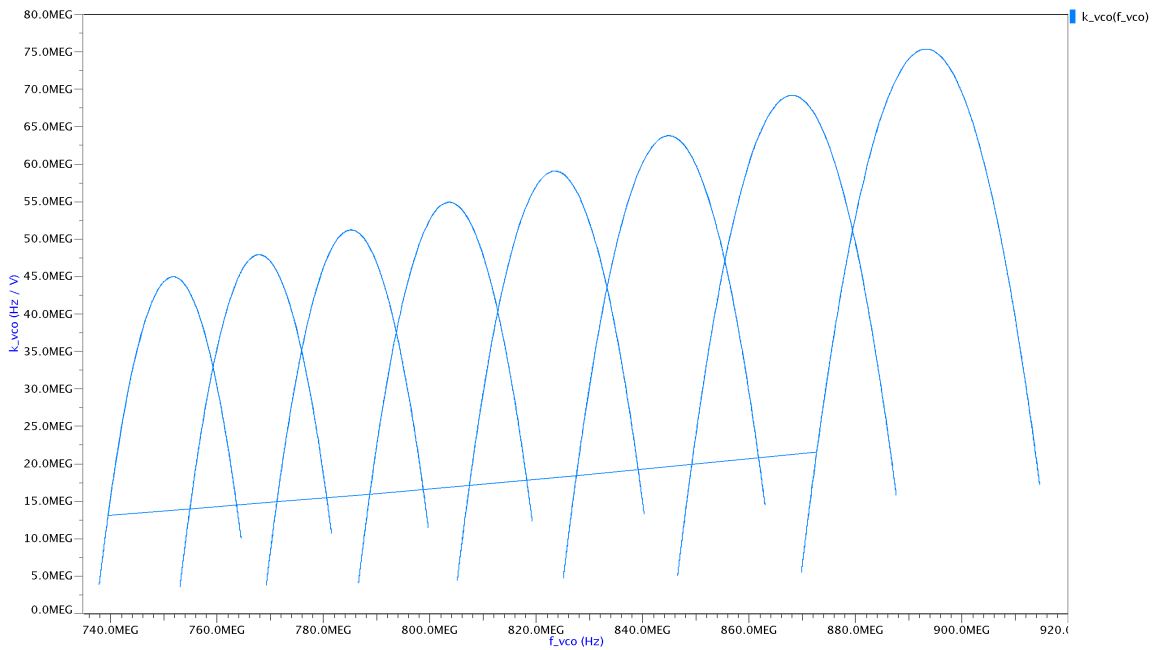
(b) $f_{vco}(v_{tune_eff})$.

Figure 4.7.: Simulation results of the test bench for the VCO model with differential digital output (vco_diff_digi_out_tb).

4.3. Modelling the Frequency Synthesiser



(c) $k_{vco}(v_{tune_eff})$.



(d) $k_{vco}(f_{vco})$.

Figure 4.7.: Simulation results of the test bench for the VCO model with differential digital output (vco_diff_digi_out_tb) (continued).

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

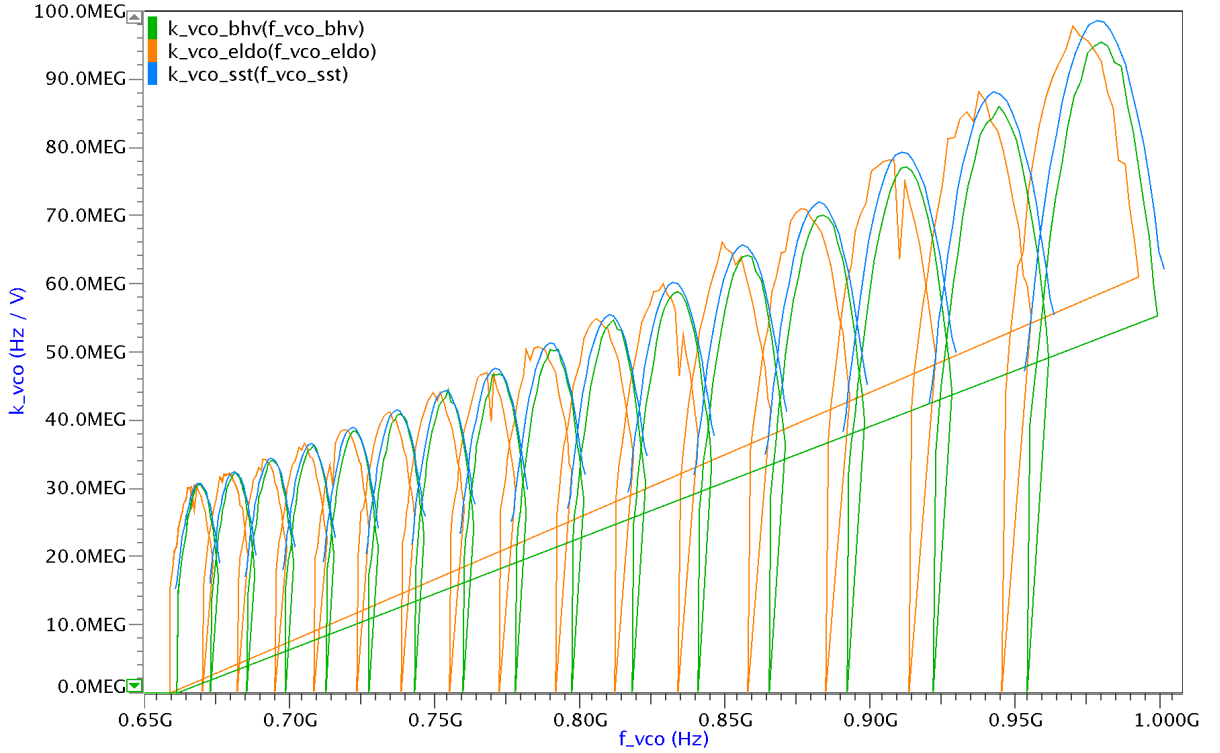
The presented simulation results for the VCO model match very well the behaviour specified in Section 4.3.2. However, to increase further the confidence in the correctness of the developed models, they have been validated against a transistor level model of an existing VCO design made available by the CSEM RF group in form of an ELDO (circuit) netlist. A thin VHDL-AMS wrapper has been implemented, which instantiates the ELDO netlist within one architecture and the `vco_diff_sine_out` behavioural model calibrated to match the specifications of the reference model in another architecture. This wrapper has been integrated in the already described VCO test bench and simulated for both architectures. The resulting $k_{vco}(f_{vco})$ curves have been superimposed in Figure 4.8. Additionally, the figure also contains the results of an equivalent steady-state analysis of the ELDO netlist. All three curves match well each other concerning the shape. They are only slightly shifted, but within an acceptable tolerance. The simulation performances summarised in the table below the figure show the huge advantage of the behavioural model with respect to the transistor level ELDO model. The simulation takes only about 6 min instead of nearly 2 h, respectively. The behavioural model simulates about 26 times faster than the ELDO model with about the same number of solution steps and dynamic behaviour. The table also shows that the steady-state analysis can deliver the $k_{vco}(f_{vco})$ relationship in even less time than the behavioural model. However, this is only the case for the behavioural VCO model with sinusoidal output that matches the output of the ELDO model. If an equivalent VCO model with discrete digital or frequency output had been used, they would give the same result in a few seconds (compare with the simulation performances presented in Table 4.3 for a similar but not identical case). Moreover, a steady state analysis does not capture the dynamic behaviour of a model, as the transient analysis does. Again, the presented approach is not only interesting for the pure validation of the behavioural model. The same approach can be used by a designer for the bottom-up verification of his transistor level implementation with the help of the test bench and behavioural model he already used during the top-down design phase to evaluate and refine the specification of the component.

4.4. Application of the RF_TRX Library to the Design of a Binary FSK Transmitter

The binary FSK transmitter used in this section as an example is based on the frequency synthesiser architecture depicted in Figure 4.2 with direct modulation done at the Σ - Δ modulator input. The Σ - Δ control words are generated by a bit-stream modulator model based on the bit stream sampled at its input, to which it can apply a pulse-shaping technique (none, ramped, or raised-cosine) [15]. The PA is not modelled in this example, as the focus is on the evaluation of the modulation process.

4.4.1. Implementation of the FSK Transmitter Model

The transmitter model is realised as a structural model (Figure 4.9) instantiating the necessary RF_TRX models not directly, but through **component** declarations provided in a **package**. All generic parameters of the component instances are forwarded through the **generic** declaration of the transmitter model. This allows the parameterisation of the whole model hierarchy from the test bench. The constants used for this purpose are declared in a separate package for each design case (i.e., target specification to implement). This has several advantages. The consistency during the evaluation of different architecture options is ensured, since constants, common to all architectures, are declared only once. Architecture-specific constants can be declared in the same package and may be based on the common ones. The constants for the back-annotated component parameters from transistor level are declared in this package,



Test bench	VCO model	Analysis type	CPU time	Steps
VHDL-AMS	VHDL-AMS	transient	00:05:35.990	5 237 162
VHDL-AMS	ELDO	transient	01:58:04.020	5 233 272
ELDO	ELDO	steady-state	00:01:59.970	384

Figure 4.8.: Validation of the behavioural VCO model with differential sinusoidal output against its circuit level implementation in form of an ELDO netlist by comparing the simulated $k_{vco}(f_{vco})$ curves. For the ELDO netlist, the $k_{vco}(f_{vco})$ curve can be obtained from a steady state analysis (number of considered harmonics $n_{\text{harm}} = 5$). The dynamic behaviour of the VHDL-AMS and ELDO models of the VCO has been validated with a common test bench sweeping the tuning voltage for each band during a transient analysis ($h_{\text{max}} = 50 \text{ ps}$, $t_{\text{stop}} = 250 \mu\text{s}$, $\text{eps} = 10^{-6}$). All simulations were done using ADMS 2008.2 on an Intel Core 2 Quad 2.66 GHz CPU, 4 MB cache, 4 GB RAM running Linux 2.6.9 x86_64.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

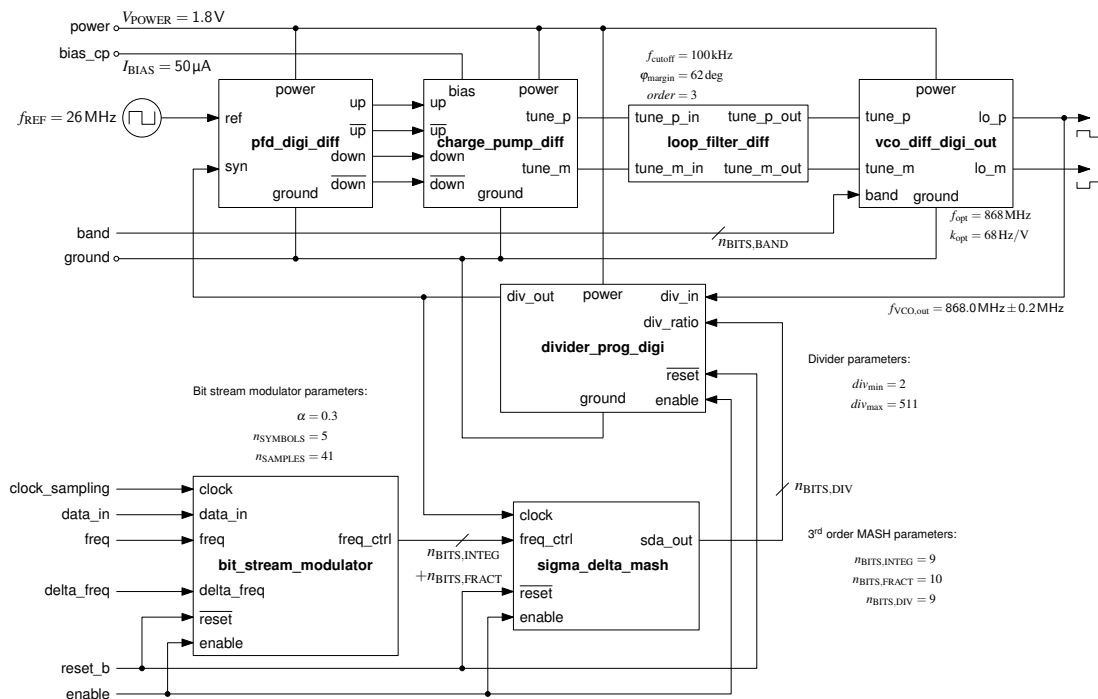


Figure 4.9.: Structure of the binary FSK transmitter model with differential digital VCO output. The schematic is annotated with the component parameters for the first design case ($f_c = 868$ MHz, $MI = 2$, $DR = 100$ kbit/s) described in Section 4.4.2.

too. The stimuli generation (in our case pseudo-random bit-stream, reset, clock, and power supply) is factorised into an own model and parameterised in the test bench by the *design case constants package*.

It is therefore very simple to create a test bench for each design case and architecture to be evaluated. The test bench just needs to include the constants package for the design case, to declare all signals to interconnect the instances of the stimuli generator and of the transmitter architecture models, and to carry out the constants mapping to the generic parameters. Finally, the mapping of the components instances is done in a **configuration** for the test bench. This allows to reuse the same structural model during the top-down design and bottom-up verification phases by selecting the appropriate architectures of the component models. As for the component test benches, the transmitter model is integrated into the automated build system with simulation control scripts and waveform display setups for each design case model configuration. This careful model organisation (Figure 4.10) achieves a full orthogonalisation of the *model structure*, *parametrisation*, and *abstraction selection* aspects avoiding code duplication as well as simplifying the addition of new design cases and model configurations.

4.4.2. Top-Down Design Exploration for Different Target Specifications

Three different design cases have been chosen to validate the FSK transmitter model and to show its flexibility. The first two examples represent possible wireless sensor network applications. The targeted output frequency (868 MHz) is in the SRD activity band and the chosen modulator data rate (100 kbit/s) allows to reach the duty cycle requirement ($< 1\%$) for a low data rate application, e.g., a sensor data transfer. The two simulations differ in the modulation index, set respectively to 2 (Figure 4.9) and 1, to analyse the influence of a frequency shift modification on modulation. The third design case targets a

4.4. Application of the RF_TRX Library to the Design of a Binary FSK Transmitter

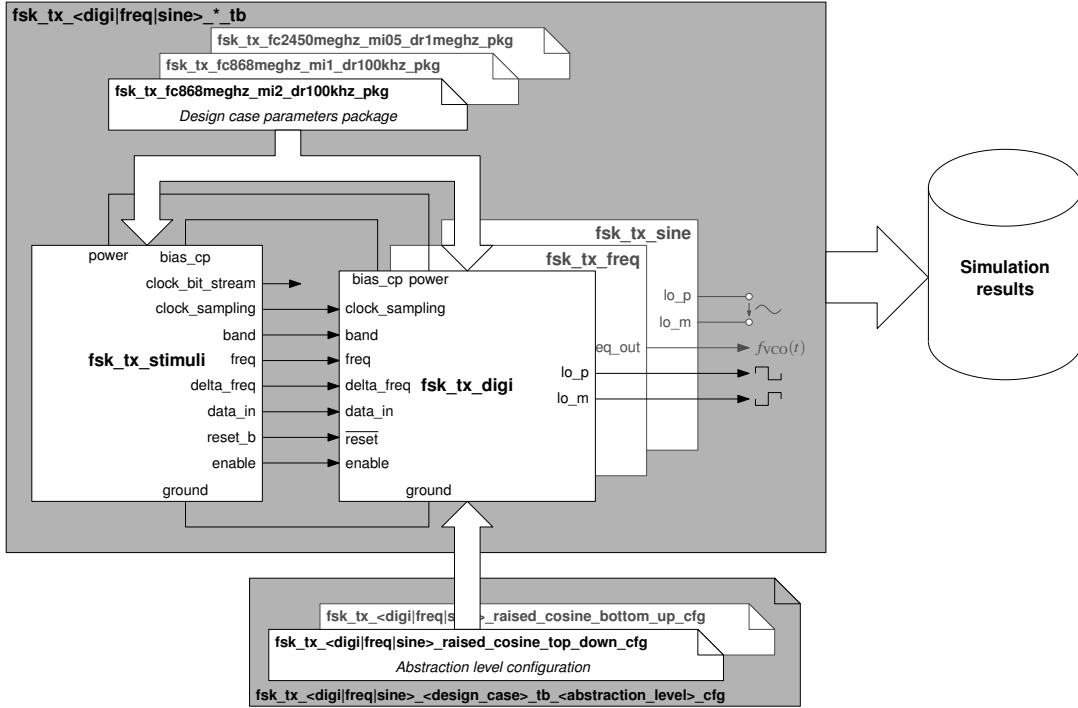


Figure 4.10.: Organisation of the FSK transmitter test benches.

transmission in the ISM band, for a compliant BluetoothTM or WibreeTM application. The chosen carrier frequency and data rate are 2.45 GHz and 1 Mbit/s, respectively, suitable for applications like audio streaming. For this scenario, a modulation index of 0.5 has been used.

Figure 4.11 shows the simulation results for the transient analysis of the first design case (Figure 4.9) using ADMS 2008.2. It shows how the random bit stream (data at clock_bit_stream rate) is oversampled by the bit modulator to transform it into a fractional division rate (freq_ctrl_real) of raised cosine pulse form, which in turn is translated by the Σ - Δ modulator into the integer division ratio (sda_out) used by the divider of the PLL. The latter divides the VCO output signal (lo_p and lo_m) into the signal div_out, which is compared with the reference signal of the quartz oscillator (ref) by the PFD. The charge pump is controlled by the PFD via the signals up, up_b, down, and down_b. Its operation yields the unfiltered tuning voltage (v_c1) for the VCO at the charge pump output over the capacitor C_1 of the loop filter. The unfiltered tuning voltage v_c1 shows a lot of noise due to the fast switching integer division ratio ordered by the Σ - Δ modulator. The filtered tuning voltage at the VCO input (v_c3) and the resulting effective tuning voltage over the varicap (v_tune_eff) show much less noise because the loop filter averages the input signal. The tuning voltage (v_tune_eff) and the resulting VCO output frequency (f_vco) follow well the shape of the random bit stream—a qualitative sign that the FSK modulation works. The VCO output frequency (f_vco) is correctly positioned around the centre frequency (freq) of the selected band (band) with the frequency difference (delta_freq) added and subtracted from the centre frequency to realise the FSK modulation of the bit stream.

In order to have indications on the modulation quality, the signal spectrum and the eye diagram are evaluated. Depending on the targeted band (SRD, Bluetooth), different requirements have to be fulfilled in order to verify that the transmitted signal respects the standard specification and does not perturb adjacent channels. For this reason, the modulated signal spectrum has to be compared to the spectral

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

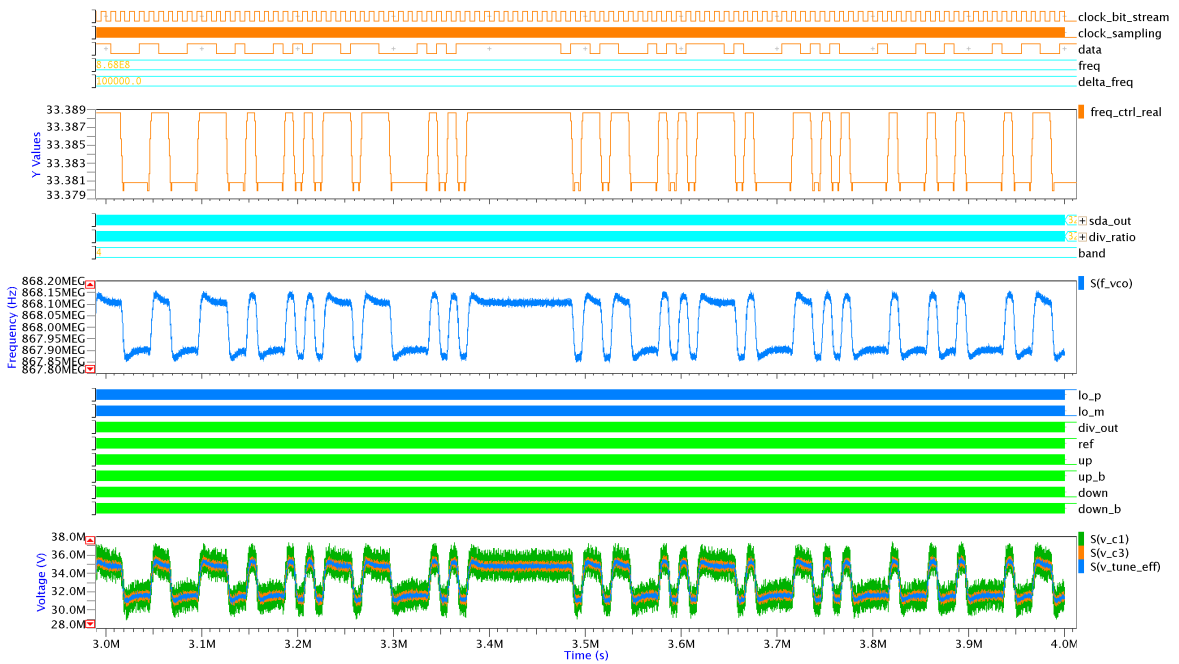
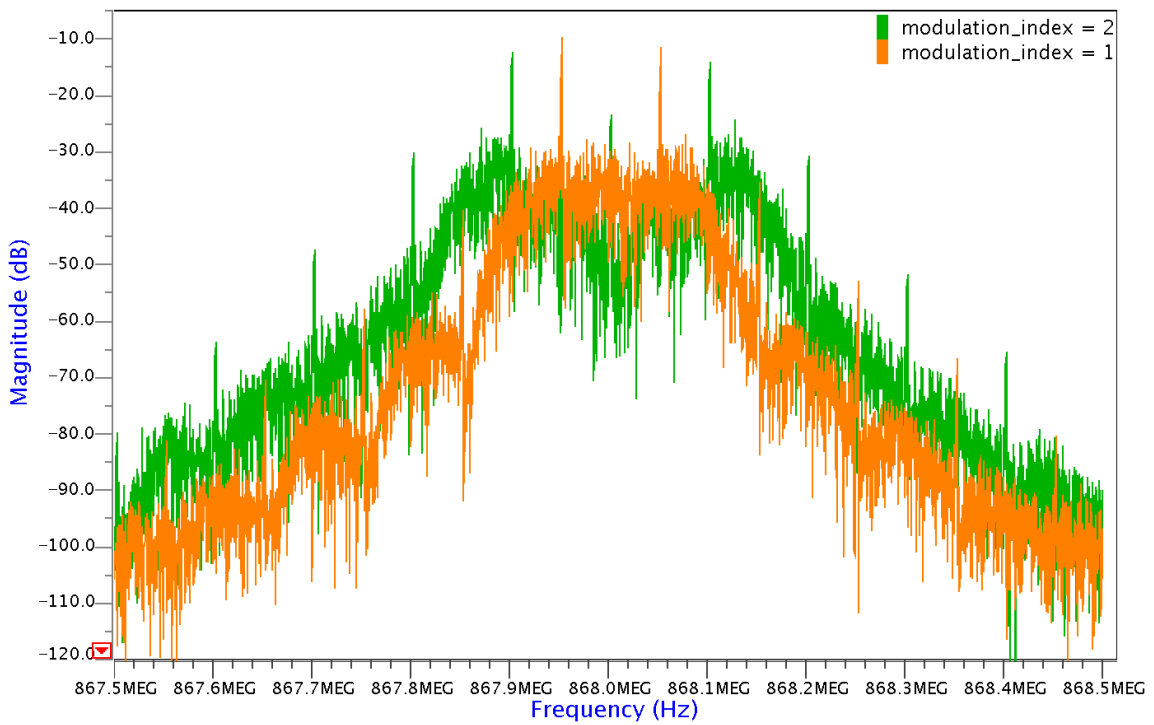


Figure 4.11.: Transient analysis of the binary FSK transmitter model with differential digital VCO output for the first design case ($f_c = 868$ MHz, $MI = 2$, $DR = 100$ kbit/s). The simulation performance is indicated in Table 4.4a.

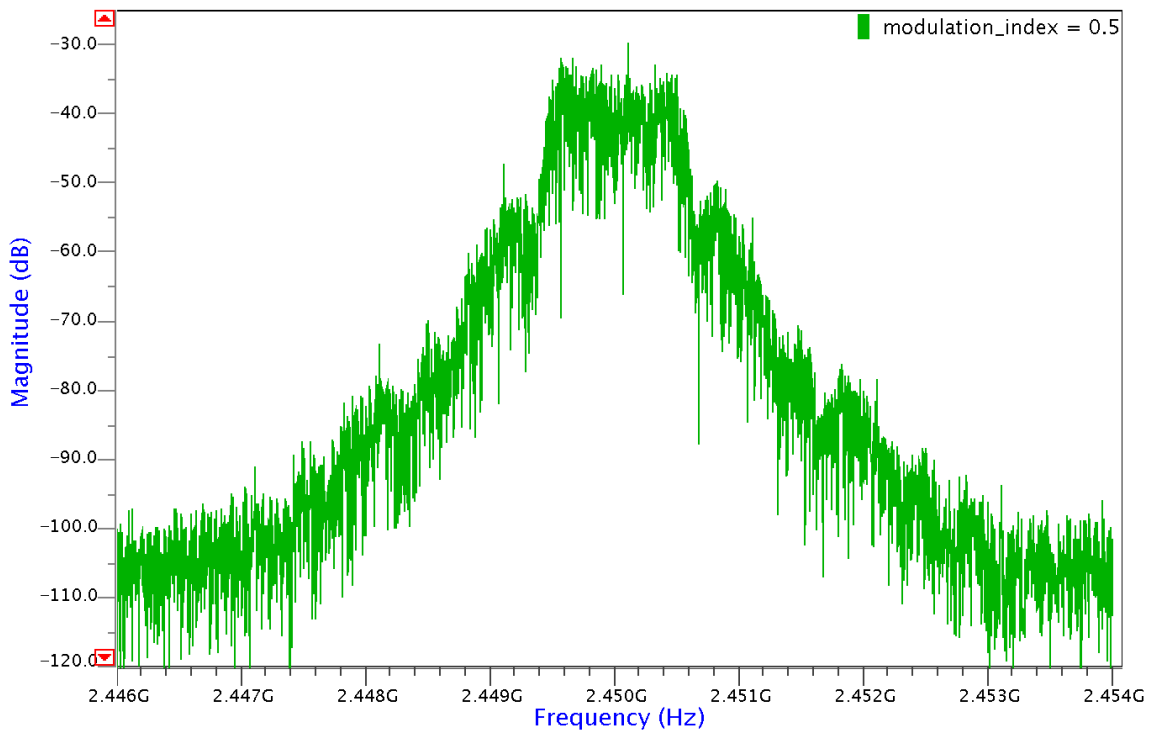
mask, verifying that it never overcomes this reference curve. In the presented simulations (Figure 4.12), only the contributions coming from the ideal PLL modulation determine the final signal spectrum. The noise generated by the PLL blocks and the PA nonlinearities are thus not taken into account. Nevertheless the modulated signal at the output of the VCO already gives a fair indication of the occupied spectrum highlighting possible problems in the modulation process or in the PLL design. By applying an ideal demodulation on the same VCO signal and processing the result, the eye diagram can be built (Figure 4.13). The study of the eye diagram allows to gather useful information on the transmitted signal and on the demodulation requirements. An example is the choice of the optimum sampling instant to minimise inter-symbol interference. Concerning the top-down design methodology, this analysis shows mainly the impact of the Σ - Δ quantisation noise on the signal integrity, giving indications on the trade-offs between the reference, the carrier, and the filter cut-off frequencies. For the bottom-up verification phase, it is very convenient that the RF_TRX models can be also parameterised with the back-annotated device parameters—especially because the device parameters calculated by the top-down architectures usually cannot be exactly implemented on the transistor level due to design rules limitations and other constraints.

Table 4.4 summarises the simulation performances obtained for the three design cases of the FSK transmitter. It can be seen that, as expected, the 1st and 2nd design cases have very similar simulation performances, as their specifications only differ slightly in the modulation index. Three transmitter model variants were developed, in order to investigate the impact of the different VCO output modelling approaches on the transmitter model simulation performance. As expected, the variant with digital VCO output performs better than those with frequency output (variant “frequential”) and with sinusoidal

4.4. Application of the RF_TRX Library to the Design of a Binary FSK Transmitter



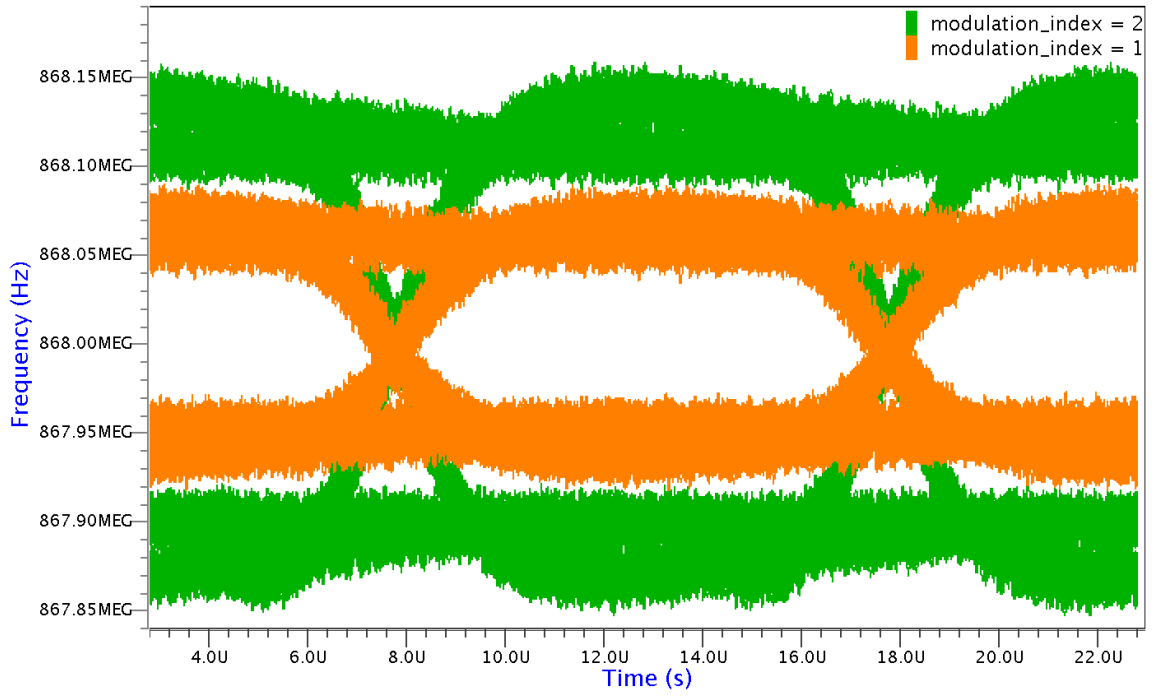
(a) $f_c = 868 \text{ MHz}$, $MI = \{2, 1\}$, $DR = 100 \text{ kbit/s}$.



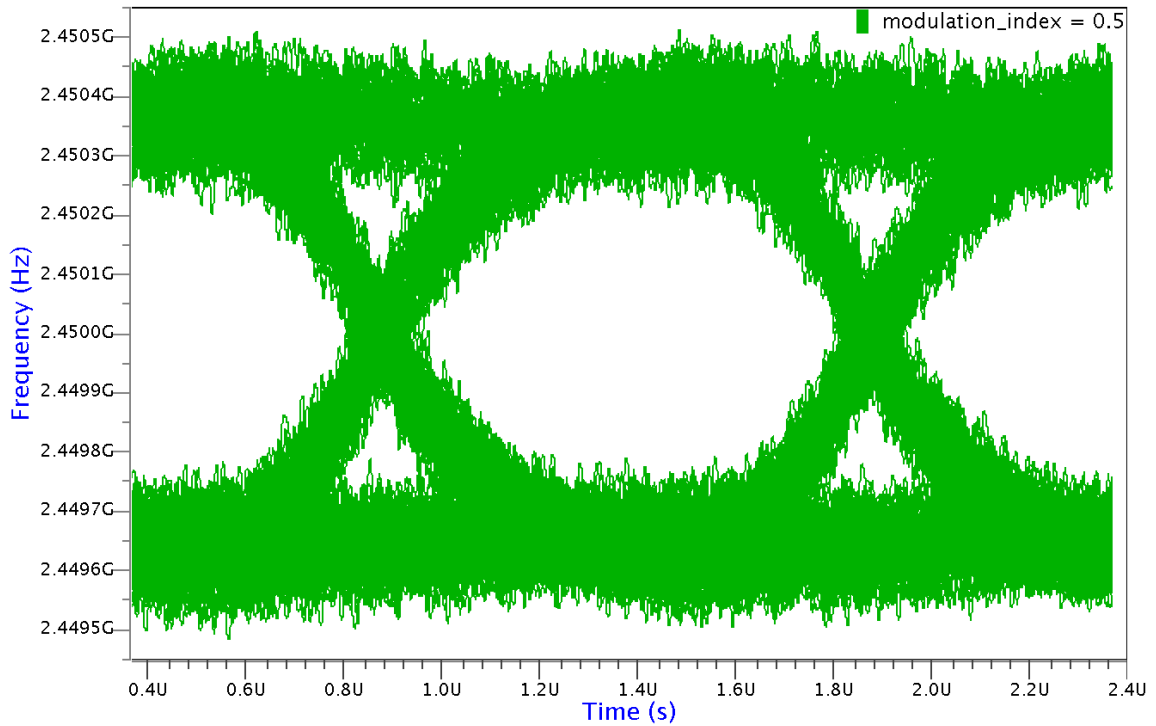
(b) $f_c = 2.45 \text{ GHz}$, $MI = 0.5$, $DR = 1 \text{ MHz}$.

Figure 4.12.: Frequency spectrum of the FSK transmitter output signal.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems



(a) $f_c = 868 \text{ MHz}$, $MI = \{2, 1\}$, $DR = 100 \text{ kbit/s}$.



(b) $f_c = 2.45 \text{ GHz}$, $MI = 0.5$, $DR = 1 \text{ Mbit/s}$.

Figure 4.13.: Eye diagram of the FSK transmitter output frequency.

4.4. Application of the RF_TRX Library to the Design of a Binary FSK Transmitter

Table 4.4.: Comparison of the simulation performances among the FSK transmitter model variants. All simulations were done with ADMS 2008.2 [116] ($\epsilon_{ps} = 10^{-6}$) on an Intel Core 2 Quad 2.66 GHz CPU, 4 MB cache, 4 GB RAM running Linux 2.6.9 x86_64.

(a) $f_c = 868 \text{ MHz}$, $MI = 2$, $DR = 100 \text{ kbit/s}$, $t_{\text{stop}} = 4 \text{ ms}$.				
VCO output	h_{max}	CPU time	Steps	Events
digital	$1 \mu\text{s}$	00:03:50.970	2 637 297	44 173 736
frequential	4 ns	00:04:41.240	3 795 161	6 154 945
sine	100 ps	01:16:38.380	42 086 453	30 285 926

(b) $f_c = 868 \text{ MHz}$, $MI = 1$, $DR = 100 \text{ kbit/s}$, $t_{\text{stop}} = 4 \text{ ms}$.				
VCO output	h_{max}	CPU time	Steps	Events
digital	$1 \mu\text{s}$	00:03:49.970	2 634 484	44 164 783
frequential	4 ns	00:04:47.050	3 789 144	6 152 387
sine	100 ps	01:17:03.080	42 087 532	30 276 913

(c) $f_c = 2.45 \text{ GHz}$, $MI = 0.5$, $DR = 1 \text{ Mbit/s}$, $t_{\text{stop}} = 400 \mu\text{s}$.				
VCO output	h_{max}	CPU time	Steps	Events
digital	100 ns	00:01:58.550	911 153	14 935 101
frequential	500 ps	00:02:24.410	1 889 599	4 145 715
sine	40 ps	00:20:32.120	11 417 132	11 016 978

differential voltage output (variant “sinusoidal”), even though in this case a larger number of discrete events need to be processed by the digital simulator. The reason is that the latter two variants require a much smaller maximum integration step size h_{max} to correctly sample all analogue signals. The presented simulation results (Figures 4.11, 4.12, and 4.13) were obtained with the variant “digital”. The “sinusoidal” variant is more precise, but much slower with respect to the “digital” variant (e.g., more than an hour of simulation time compared to ca. four minutes for case (a) in Table 4.4), while its spectra and eye diagrams are very close to those of the “digital” variant. This sufficient precision and high simulation performance makes the RF_TRX library models very useful for the top-down architecture exploration and bottom-up verification phases.

4.4.3. Bottom-Up Verification of a Design Case Implementation

As described in Section 4.4.1, the structural model of the FSK transmitter used for the top-down architecture exploration can be reused for its bottom-up verification. The configuration just needs to be switched to use the architectures of the component models targeted for the bottom-up phase and the component parameters from the transistor level implementation need to be back-annotated into the design case’s parameter package. The simulation performances for the purely behavioural system won’t dramatically differ between the top-down and bottom-up variants, as long as the implemented dynamic behaviour does not differ but only the way of the initialisation of the secondary parameters during the elaboration phase. This has been already shown in the case of the VCO model in Table 4.3.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems

However, the bottom-up verification can be taken a step further by verifying the transistor level implementation of individual components in the system context using mixed-level simulations. The approach is the same as the one used for the validation of the behavioural VCO models against a transistor level reference model described in Section 4.3.4. The transistor level implementation in form of an ELDO netlist and the corresponding behavioural model of the component are instantiated in two separate architectures of a thin wrapper, which adapts the netlist's interface to the interface expected by the system model and calibrates the behavioural model of the component to match the behaviour of the component's transistor level implementation. To verify the calibration of the behavioural component model, the test benches available for each individual component can be reused. The wrapper is then integrated into the system model and the configurations and parameter package for the design case are adapted. Then, the system model can be simulated either using exclusively the behavioural models or using the transistor level models for selected components by simply switching the architecture of the corresponding wrapper in the configuration of the system model.

For the first design case of the FSK transmitter ($f_c = 868$ MHz, $MI = 2$, $DR = 100$ kbit/s, cf. Section 4.4.2), this mixed-level bottom-up verification has been performed for the two key components of its PLL: the CP and the VCO. Figure 4.14 shows the influence of the different CP and VCO wrapper architectures (either *behavioural* or *ELDO*) on the FSK transmitter output spectrum calculated from the VCO's sinusoidal output signal after a transient analysis for 4 ms using ADMS 2008.2. It can be seen that the principal characteristics (overall shape, position and amplitude of the harmonic peaks, etc.) of the FSK output spectrum are correct for all four presented variants especially within the frequency range defined by the second-order harmonic peaks (867.8 MHz to 868.2 MHz). Outside this frequency band, the two variants using the ELDO netlist model of the VCO show a higher noise level. This is no surprise, as the ELDO models contain much more details. It is interesting to note that using also the ELDO architecture for the CP model lowers the noise floor compared to the variant, in which all components, except the VCO, are represented by behavioural models. The transistor level simulation of the CP model inside the FSK transmitter also showed how important it is to implement the primary non-idealities into the behavioural component models of the RF_TRX library. Wouldn't the PFD model offer a parameter to specify the *reset delay* of its two D-flip-flops to match the delay of the switching transistors in the CP imposed by the technology, the PLL of the FSK transmitter would show a stability problem: The steering pulses at the CP's control inputs would become too short for the switching transistors to correctly react. The outcome would be the rejection of the transistor level implementation due to an unacceptable noise level in the FSK modulation, even though it is correct and just because the behavioural models used in the bottom-up verification were too ideal.

To conclude, let us have a look on the simulation performances of the four mixed-level simulation variants presented in the table below the output spectra in Figure 4.14. The fully behavioural system model of the FSK transmitter simulates within only about 1.5 h. Please remember that we had to choose the VCO model with sinusoidal differential voltage output in order to match the interface of the ELDO netlist of the VCO implementation. Using a behavioural VCO model with differential digital output for this design case would yield a simulation time of only about 4 min (Table 4.4a), i.e., nearly 24 times faster! Once the CP is represented by its ELDO netlist, the simulation takes already half a day. Replacing only the VCO model with its ELDO netlist slows down the simulation to take nearly a full day. Finally, using the ELDO netlist for both components requires the designer to wait two days for the simulation results. These simulation times are still acceptable for a *sign-off verification* of the transistor level implementation of a component during the bottom-up design phase. To put these figures further into perspective, a transistor level simulation of the whole PLL (*without* $\Sigma\text{-}\Delta$, bit stream modulator, and quartz oscillator blocks) has been tried. For only a 10 μ s transient simulation, the overall elapsed

CPU time exceeded 24 h and after termination it was not possible to access the simulation results. This impressively shows that transistor level verification of an RF system design is not practical, at least with today's available workstations. It demonstrates again the importance of abstracting away the details, which do not have a major impact on the system level behaviour, in order to considerably speed up system level simulations.

4.5. Conclusions and Outlook

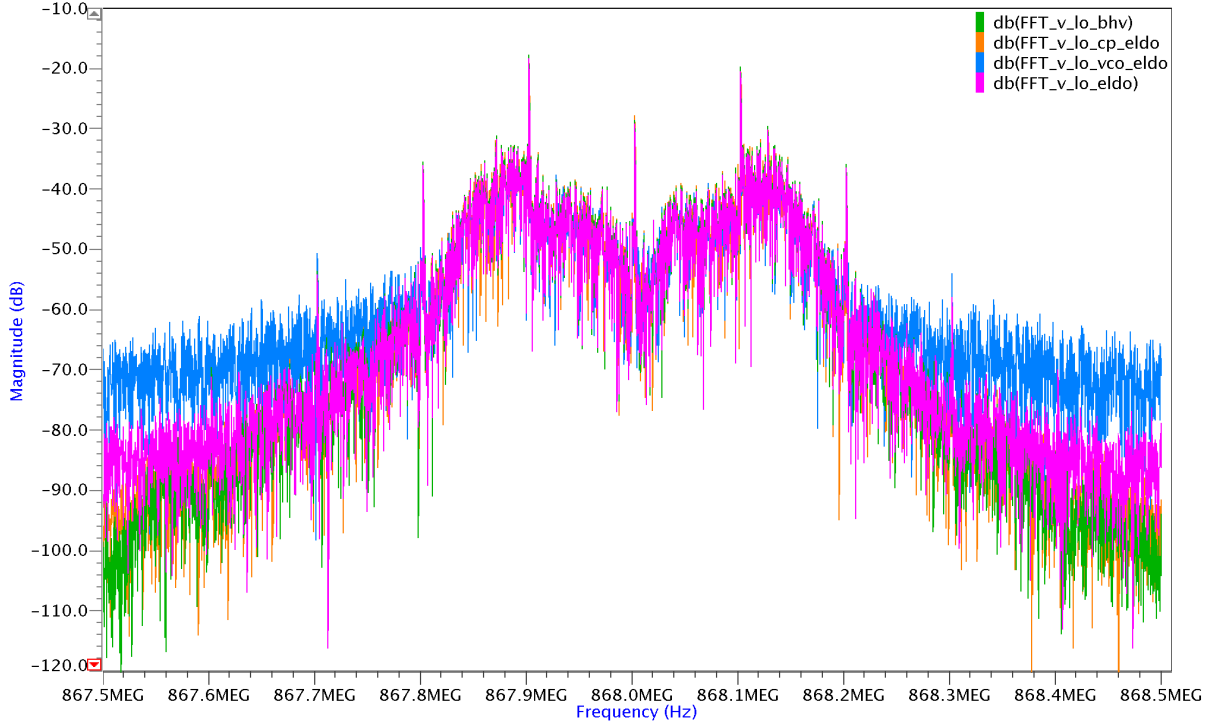
This chapter presented a methodology for modelling a complex RF system, namely the frequency synthesiser and the modulation for the transmitter part of a binary FSK transceiver. In a first step, the behaviour of the principal system components has been specified. Then, the corresponding models have been individually designed, implemented, and tested. It has been demonstrated how the different needs (especially concerning parameterisation) of the top-down and bottom-up design phases can be captured in the same VHDL-AMS model by means of a common pin-accurate interface (**entity**) and dedicated **architectures** for the different implemented abstraction levels. The systematic usage of **component** instantiation and **configurations** for structural models ensures their reusability throughout the design process and assures consistency when simulating different abstraction levels of the same model. The key component models have been validated against transistor level reference models. It has been shown how the elaborate test benches developed for the behavioural component models in the RF_TRX library can be reused by the RF designer for the top-down design and bottom-up verification of the transistor level implementation of a component to meet a concrete specification.

Finally, the full system model of the FSK transmitter has been assembled, thoroughly tested, and the simulation results presented. Its careful organisation allowed to achieve a full orthogonalisation of the *model structure*, *parametrisation*, and *abstraction selection* aspects avoiding code duplication as well as simplifying the addition of new design cases and model configurations for top-down architecture exploration. It can serve as a template for the implementation of other complex system level test benches. A systematic approach for the individual validation of transistor level component implementations in the system context has been presented, which achieves considerable gains in simulation performance compared to a full transistor level verification of the system.

The development of the VHDL-AMS model library RF_TRX helped to establish best practices regarding the communication between model developers and RF designers as well as the organisation and documentation of the models. Their application ensures a maximum flexibility, reusability, validity, and maintainability of the models. This approach shows that a complex RF system can be simulated rapidly and precisely by making the right abstraction choices. The results demonstrate impressively that such VHDL-AMS system simulations can provide the RF designer essential information, e.g., related to the spectral density or the eye diagram that transistor level simulations cannot give in such a short time without sacrificing too much precision.

Future working directions include the consideration of different noise forms and the modelling of power consumption in the models as well as the development of new component models for the library to increase its coverage in the long term on the whole transceiver chain. However, for a complex heterogeneous SoC, the transceiver will constitute a single component. Therefore, even the behavioural modelling approach demonstrated in this chapter will show its limitations in terms of simulation performance and modelling capabilities of "classical" HDLs. New modelling formalisms supporting higher levels of abstractions are needed, which are discussed in the next chapter.

4. A VHDL-AMS-Based Methodology to Efficiently Model RF Systems



CP model	VCO model	CPU time	Steps
VHDL-AMS	VHDL-AMS	01:34:41.080	47 430 482
ELDO	VHDL-AMS	12:11:33.250	51 190 531
VHDL-AMS	ELDO	23:24:22.100	47 338 198
ELDO	ELDO	38:44:06.970	51 243 167

Figure 4.14.: Bottom-up verification of the CP and VCO circuit level implementations selectively integrated into the FSK transmitter model with differential sinusoidal VCO output. For each case, the frequency spectrum of the FSK transmitter output signal has been calculated from a transient analysis ($h_{\max} = 100$ ps, $t_{\text{stop}} = 4$ ms, $\epsilon_{\text{ps}} = 10^{-6}$) with ADMS 2008.2 on an Intel Core 2 Quad 2.66 GHz CPU, 4 MB cache, 4 GB RAM running Linux 2.6.9 x86_64. The simulation performances are compared for the CP and VCO being modelled either with VHDL-AMS or ELDO.

5. Enhancing the OSCI SystemC AMS extensions for Efficient Multiphysical Systems Modelling

This chapter introduces new modelling capabilities on top of the recently standardised OSCI SystemC AMS extensions to describe energy conserving multiphysical systems in a formal and consistent way at a high level of abstraction. In a first step, a technique is presented to integrate the measurement units associated to variables and parameters as an integral part of their *quantity data type*. This enforces correct model assembly through strict interfaces and coherent formulas describing the analogue behaviour by means of *dimensional analysis* implemented by these quantity datatypes. To ensure the reusability of models using these new quantity datatypes, their interfaces need to become parameterisable by applying special coding techniques. This has been demonstrated on a library of block diagram modules for the TDF MoC. Using all these techniques, a completely new Bond Graph (BG) MoC has been integrated into the SystemC-AMS PoC simulator. It implements the bond graph formalism to describe multiphysical systems in a more adapted and generic way than it is currently possible with the standard MoCs of SystemC-AMS. With the help of causality analysis, bond graph models are transformed at elaboration time into an equivalent signal flow model to achieve a high simulation performance at system level. Therefore, the BG MoC can support as a side product block diagram descriptions in parallel to bond graphs. An extensive library of basic bond graph and block diagram primitives is proposed and can be augmented by user-defined primitives. The causality analysis of the bond graph models at elaboration time gives the designer insight into the computational structure of his models and enables other formal checks, which provide him with hints regarding modelling problems such as algebraic loops, multiple drivers, and ill-formed models.

5.1. Introduction

The previous chapter showed that “classical” AMS HDLs reach their limits in terms of simulation performance and modelling capabilities for complex heterogeneous systems. Their design requires the cooperation of different domain experts, who employ different modelling formalisms and tools. However, to understand the functional interaction of the different system parts involving potentially different physical domains and thus to be able to refine the system architecture and to derive consistent component specifications, a *system model* needs to be created early on in the design process and continuously refined from functional to architectural abstraction levels. Thus, an *executable specification* is created, which later helps to verify the correct integration of the individually designed components to a system. The need for such a “golden” system model is rising with the system’s heterogeneity. A single person is no more able to grasp all the system’s details and their impact on its global behaviour. The challenge moves thus from the design of the individual components, which is nowadays within one engineering discipline usually well understood and supported by tools, to the *integration* of the individually validated components into a complete system on the so-called *Electronic System Level* (ESL).

A successful integration of heterogeneous parts requires a strict specification of the interfaces between the system components, which should be expressible in the system model. This includes not only the *data type* (storage format) of the *value* of a *quantity* exchanged through ports or used to parameterise

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

the component, but also the semantical information on how the value is to be interpreted. The latter is expressed by the *measurement unit*, which itself expresses the abstract *dimension* of measure and the actual *system of units* used to measure the amount of the quantity. Their usage is common practice in any natural science and engineering field for hand calculations and written documentation. It permits to check the coherency of the equations used to describe the system by means of *dimensional analysis*. However, most common modelling and programming languages permit only to specify the data type of the value and not of the whole quantity during the declaration of constants, variables, ports, and parameters. They primarily provide means to manage the complexity of a system by structuring the system and the treated information in a hierarchy and to express their processing in terms of control structures and computations. This is less a problem in single domain systems designed primarily by one engineering discipline, as the correct interpretation of the data as a quantity can be supported by using a consistent naming scheme based on established symbols or acronyms and by using always the same measurement unit for a dimension and annotating its usage in the comments. Manual consistency checks are then facilitated. These measures lower the risk of interconnecting wrongly the system components and implementing incoherent equations describing the system behaviour, which lack, e.g., a conversion factor or accidentally sum up quantities of different dimensions.

However, in multiphysical systems the risk for misinterpreting a quantity just based on its identifier is rising due to partially conflicting standards for quantity symbols (e.g., v for voltage and speed) in different engineering disciplines and different common practices regarding the units (e.g., feet and meter for a distance) and scale factors (e.g., μm) to measure them. This can lead to very hard to spot problems, which may stay undiscovered as the simulation results seem to be meaningful and in the expected order of magnitude. This is especially true when integrating IPs from different sources and reusing legacy models with potentially diverting specifications requiring an adaption of the interfaces. For example, one of the root causes for the loss of NASA's Mars Climate Orbiter in 1999 during the orbit entering manoeuvre was a forgotten unit conversion between imperial and SI units between different programmes used for the course corrections [127]. The consistent usage of a single system of units might have prevented this.

However, the parallel usage of different systems of units in a single model might be necessary for good reasons. This can be, e.g., illustrated with the VHDL-AMS model of a micromechanical yaw rate sensor system presented in Mähne et al. [108]. The reduced-order model of the sensor was extracted from an FEM model of the micromechanical structure, which used for numerical reasons the μMKS system of units (displacements in μm , forces in μN , etc.). The electrical part of the system model responsible for the control of the mechanical structure's movements used the SI system of units. Therefore, unit converters at the subsystem boundaries had to be inserted. Although VHDL-AMS allows to identify different disciplines by declaring proper natures, the quantities related to such natures are still subtypes of `real`. Thus, the unit converters risked to be incorrectly implemented without the compiler noticing it.

AMS hardware description languages such as VHDL-AMS [76] or Verilog-AMS [2] have limitations regarding the support of dimensional analysis. For example, VHDL-AMS only offers a way to annotate quantities with their units for presentation purposes:

```
1 subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
2 attribute UNIT OF VELOCITY : subtype is "meter/second";
3 attribute SYMBOL OF VELOCITY : subtype is "m/s";
```

The full support of dimensional analysis was considered during language design, but rejected as the required changes to the type system would have rendered it incompatible with VHDL [12].

Modelica [121] allows to annotate the type **Real** with its dimension and SI measurement unit but unfortunately not the numeric constant assigned to it:

```
1 Real(unit="m.s-1") v = 2.0;
```

It depends on the tool if this information is used to check for dimensional and unit consistency, e.g., Dymola [40] and SimulationX [83]. Broman, Aronsson, and Fritzson [25] describes an extension of the Modelica language and a prototype implementation for improved dimensional inference, unit checking, and declaration of new (non-SI) units.

In the proprietary SimscapeTM language [114], introduced by The MathWorks, Inc., in 2008 for multidomain physical systems modelling on their MATLAB/Simulink products, declaration members such as parameters, variables, inputs, and outputs are represented as a value with associated unit:

```
1 variables
2     w = { 0, 'rad/s' };
3 end
```

The unit is represented as a string. The individual units need to be defined in the unit registry either as the fundamental unit of a new dimension or as a scalar or affine conversion of another unit:

```
1 pm_adddimension('length', 'm');
2 pm_addunit('cm', 0.01, 'm');
3 pm_addunit('N', 1, 'kg*m/s^2');
4 pm_addunit('Fh', [5/9 -32*5/9], 'C');
```

Based on this information, the Simscape unit manager is capable to do and check unit conversions at run-time imposing a certain performance penalty.

The functional programming language F# [98] makes units declaration and dimensional analysis integral part of the language:

```
1 [<Measure>] type kg
2 [<Measure>] type N = kg m/s^2
3 let gravity = 9.808<m/s^2>
4 let metresToFeet (l:float<m>) = 1 * 3.28084<ft/m>
```

It checks all expressions using units for consistency at compile time without causing any run-time penalty. One limitation is that it does not support dimensions (classes of units, e.g., mass).

The three approaches [98, 114, 121] make units part of their language syntax and implement dedicated support for dimensional analysis in the compilers and other development tools facilitating improved error detection and reporting. However, the mentioned languages have their limitations in the modelling of the software/hardware interaction, in the support of dedicated MoCs for heterogeneous system modelling, and in the reuse of legacy code and models, which require the usage of a very flexible and extensible simulation framework. This is the strength of the C++-based open source simulation framework *SystemC* [20, 66, 124], which classes and methods support the description and simulation of digital hardware/software systems from functional down to register transfer level by using the *Discrete Event (DE) Model of Computation (MoC)*. It has seen wide industry adoption over the past decade. Its development and standardisation is coordinated by the OSCI consortium. Since 2006, it is an IEEE standard [79]. The openness of this environment facilitates the integration of other libraries and legacy code and allows the implementation of new modelling formalisms based on dedicated MoCs, e.g., Synchronous Data Flow (SDF) or Finite State Machine (FSM) [137]. The increasing integration of AMS components into embedded HW/SW thus lead naturally to several parallel research efforts to augment SystemC with new MoCs implementing continuous-time modelling capabilities, namely SystemC-AMS [170], SystemC-A [91], and SystemC-WMS [129]. The individual properties of these different approaches have been already discussed in Section 2.3.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

In the scope of the presented research work, the author of this thesis actively contributed as member of the Open SystemC Initiative (OSCI) AMS Working Group (AMSWG) [132] to the standardisation of AMS extensions to SystemC to foster their industry acceptance. This continuing effort founded on initial work by the SystemC-AMS study group [164]. Based on the collected requirements and use cases [48], the AMSWG developed a Language Reference Manual (LRM) [130], which became in March 2010 an official OSCI standard [131]. In parallel to the standard release, the AMSWG published a user's guide [14] and Fraunhofer IIS/EAS, as one of its members, a PoC implementation of the SystemC AMS extensions 1.0. In their first version, they primarily address the needs for describing the continuous-time behaviour of purely electrical AMS SoCs by proposing three MoCs, which allow their description on different levels of abstraction using *Timed Data Flow* (TDF), *Linear Signal Flow* (LSF), and *Electrical Linear Network* (ELN). This makes them well-suited for the design of communication systems, which analogue front ends are tightly coupled to complex digital control [46], and for DSP applications. However, their modelling capabilities are not yet well suited to describe *energy-conserving multiphysical system* components with *nonlinear behaviour* in a *formal and consistent way* at a *high level of abstraction*. Its requirements specification [48] already mentions these needs to enable their usage, e.g., in the automotive sector.

This thesis work proposes new modelling capabilities to the OSCI SystemC AMS extensions to address these requirements. This chapter presents the results. First an overview on the standardised SystemC AMS extensions will be given in Section 5.2 to show its central concepts, its modelling capabilities, and implementation philosophy. Then, Section 5.3 discusses how a multiphysical system can be modelled on successively higher abstraction levels. It shows how the loss of semantic information due to the usage of more abstract and generic modelling primitives can be compensated by conserving the link to the physical domain through the annotation of the model's signals and parameters with their measurement units. The section presents more extensively the bond graph formalism as an interesting compromise between energy-conserving domain-specific modelling and generic non-conservative block diagram modelling. Bond graphs enable the unified description of the energy conserving parts of heterogeneous systems with the help of a small set of modelling primitives parametrisable to the physical domain. The resulting models have a simulation performance comparable to an equivalent signal flow model. Section 5.4 describes how quantity types and dimensional analysis can be integrated with the SystemC AMS extensions to achieve the stated goal. The proposed implementation is based on the Boost.Units library, which employs template metaprogramming techniques. Thus, the C++ compiler can check the correct assembly of the components and the coherency of the equations by means of dimensional analysis. A dedicated filter for the measurement units data types has been implemented to simplify the compiler messages and thus facilitate the localisation and interpretation of unit errors and is described in Section 5.4.2. The stricter model interfaces imposed by the dimensional analysis require new ways to write reusable models. Not only their behaviour but also their interface needs to be parameterisable in a well-defined manner. The enabling implementation techniques are proposed by the author in form of the developed SystemC AMS extensions eXperiments (SCAX) library. Its implementation builds on top of Fraunhofer's SystemC-AMS PoC implementation. The structure of the SCAX library is presented in Section 5.5. This is followed in Section 5.6 by the description of the implementation of a library of generic block diagram component models for the TDF MoC of the SystemC AMS extensions, which apply these techniques. As an application example, an electromechanical transducer driving a micromechanical resonator is modelled using this library in Section 5.6.2. The example illustrates well the limitations of the currently in SystemC-AMS available MoCs for multiphysical system modelling. Building on these experiences, Section 5.7 then presents how a completely new Bond Graph (BG) MoC has been implemented as part of the SCAX library and how it tightly integrates with the other MoCs

of SystemC-AMS. This new MoC implements the bond graph formalism to describe multiphysical systems in a more adapted and generic way than it is currently possible with the standard MoCs of SystemC-AMS. Its energy conserving modelling primitives can be parameterised to user-defined physical domains (Section 5.7.3.2). With the help of causality analysis, bond graph models are transformed at elaboration time into an equivalent signal flow model to achieve a high simulation performance on system level (Section 5.7.4). Therefore, the BG MoC can support block diagram descriptions in parallel to bond graphs. An extensive library of basic bond graph and block diagram primitives is proposed and can be augmented by user-defined primitives (Section 5.7.3.3). Several application examples are presented in Section 5.7.5 together with their simulation results to illustrate the usage of the BG MoC. The electromechanical transducer example is remodelled in Section 5.7.5.1 using the new BG MoC to show its advantages over the TDF MoC for multiphysical modelling. Section 5.7.5.2 shows how tightly the new BG MoC can interact with the DE MoC and TDF MoC of SystemC-AMS using a simple velocity sensor model for a mechanical resonator as an example. The causality analysis of the bond graph models at elaboration time gives the designer insight into the computational structure of his models and enables other formal checks, which provide him with hints regarding modelling problems. It is topic of Section 5.7.5.3. Finally, some conclusions are drawn and an outlook is given in Section 5.8.

5.2. Overview on the OSCI SystemC AMS extensions

The main goal of the SystemC AMS extensions [14, 48, 131] is to support the design of complex heterogeneous HW/SW systems by providing means to efficiently evaluate different architecture options with an executable system model of the specification. During the V-shaped top-down/bottom-up design process, the system model is continuously refined to derive and later validate the specifications for the different system components and study their interaction. In this context, the AMS extensions intend to bridge the gap between system level functional tools such as Simulink[113] or Ptolemy II [49] and mixed-signal hardware description languages such as VHDL-AMS [76] or Verilog-AMS [2]. To that end, they consider three modelling formalisms with different semantics:

Signal flow models define the behaviour of continuous-time systems as mathematical relations between quantities that represent real-valued functions of an independent variable, usually the time. The directed graph constitutes the underlying principle of signal flow modelling. Each edge represents a quantity and each vertex represents a relation. Relations may take the form of implicit or explicit equations. Thus, the simulation of signal flow models requires the resolution of DAEs. Signal flow models are non-conservative and thus do not model the conservation of energy, which is one main characteristic of physical continuous-time systems. Still, they provide an appropriate level of abstraction for AMS architecture modelling.

Dataflow models define the behaviour of data processing systems as processes communicating through (un)bounded buffers. Data is represented as streams of tokens that processes consume as inputs and produce as outputs. Dataflow models are untimed and can have different semantics. For example, *synchronous dataflow semantics* supports the modelling of single-rate and multi-rate behaviours of most data processing functions (e.g., adders, multipliers, decimators, interpolators, decoders) and allow to statically determine the order of execution of processes. It is possible to add time semantics to synchronous dataflow models to make them discrete-time models. Such modified synchronous dataflow semantics is called *timed dataflow semantics* in the context of the SystemC

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

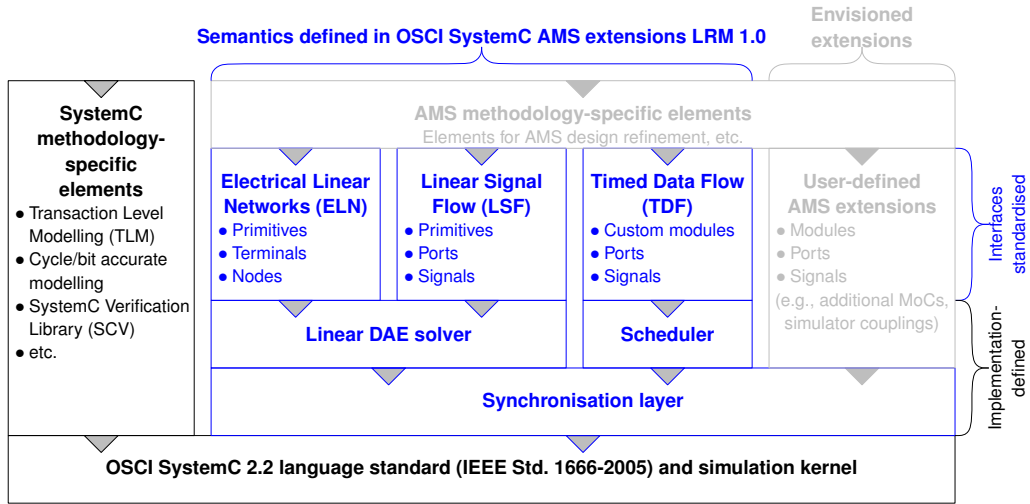


Figure 5.1.: Architecture of the OSCI SystemC AMS extensions 1.0 standard [131]. The white paper [65] and requirements specification [48] for the OSCI SystemC AMS extensions envision the future definition of a standardised API to the synchronisation layer to integrate user-defined MoCs and solvers as well as AMS methodology-specific elements in form of libraries, which will build upon the interfaces defined in the standard.

AMS extensions. Furthermore, it is possible to interpret the tokens as real-valued data samples making such models abstractions of the above defined continuous-time signal flow models.

Electrical linear networks define the behaviour of conservative continuous-time systems, such as loads, protection circuits, and buses at high frequencies, as linear network macro models based on simple electrical resistor, inductor, capacitor, and controlled source primitives. The simulation of electrical linear networks requires the resolution of DAEs in the same way as in electrical circuit simulators. However, the restriction to simple linear primitives allows to significantly reduce the demand in computing resources.

Figure 5.1 shows how these three formalisms are seamlessly integrated with the discrete event modelling formalism of SystemC by implementing the AMS extensions in a layered architecture on top of the standard SystemC kernel. Each formalism requires its own execution layer that implements the formalism's underlying Model of Computation (MoC). For the Linear Signal Flow (LSF) and Electrical Linear Network (ELN) MoCs, a linear DAE solver is required and for the multi-rate Timed Data Flow (TDF) MoC, a scheduler. A synchronisation layer coordinates the parallel execution of the different continuous-time MoCs and offers to them a common interface to interact with each other and with the DE simulation kernel of SystemC. Thus, the simulation of heterogeneous SystemC models is possible, which employ in parallel different MoCs. The LRM of the AMS extensions just defines the user interfaces to the three offered MoCs in form of the proposed modelling primitives and their semantics. It also specifies the exact synchronisation semantics between the different MoCs. The interfaces to the model execution and synchronisation layer are not standardised and can thus vary between different implementations of the standard. At the time of the writing of this thesis, one Proof of Concept (PoC) implementation of the OSCI SystemC AMS extensions is available from Fraunhofer IIS/EAS as open source under the Apache license and is called SystemC-AMS [53].

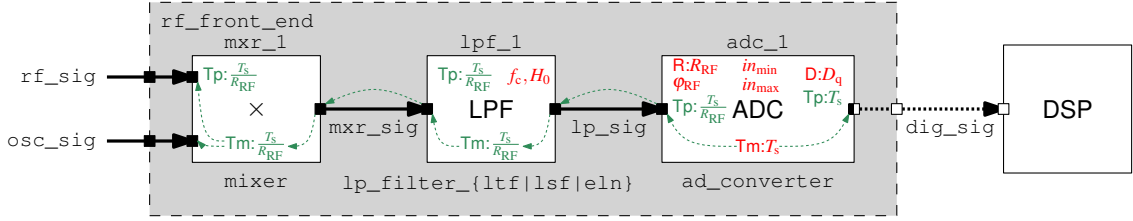


Figure 5.2.: TDF model of a simple RF front end connected to a DSP. The parameters of the modules, e.g., the cut-off frequency f_c and the gain H_0 of the low-pass filter, are denoted in **red italics**. At elaboration, the module time step (T_m) assigned to the ADC is propagated to all connected TDF modules via the TDF ports, which is indicated by the **seagreen dotted arrows**. The ratios of the port rates (R) influences the assigned port time steps (T_p) and module time steps (T_m). The output port of the ADC has a delay (D) assigned. The graphical notation is further detailed in Table A.3 on page 153

5.2.1. Timed Data Flow Model of Computation

Timed Data Flow (TDF) models consist of a set of interconnected TDF modules. A TDF module is a primitive module defining some elementary signal processing behaviour by reading one (*single-rate*) or more (*multi-rate*) data samples from its input port(s) and writing one or more data samples to its output port(s). The number (*rate*) of read/written data samples for each port per module activation is constant and needs to be configured during the elaboration of the model. Each data sample is tagged with a time stamp using a fixed time step and taking the port's configured data rate into account.

TDF modules are connected via TDF signals through TDF ports. Interconnected TDF modules form a TDF cluster. A complete AMS model may include several TDF clusters. TDF modules can interact with SystemC's discrete-event signals via special converter ports. A loop of interconnected TDF modules must specify a delay of one or more data samples at one port to achieve causality so that the cluster can become executable. The modules in a TDF cluster can be statically scheduled prior to simulation and are executed using a fixed time step during the first delta phase of the SystemC simulation cycle [79].

To illustrate the use of the TDF MoC and its multi-rate capabilities, the simple RF front end depicted in Figure 5.2 will serve as an example. The RF chain consists of a mixer, a low-pass filter, and an ADC, which digitises the downconverted RF signal for further treatment by a DSP block.

The mixer block is modelled as a single-rate TDF module. Its implementation is given in Listing 5.1 and shows the principal elements of a TDF module. The new language constructs introduced by the SystemC AMS extensions can be recognised by their `sca_` prefix. All constructs specific to the TDF MoC are defined in the namespace `sca_tdf`. A new TDF module is defined by publicly inheriting from `sca_tdf::sca_module`. Its input and output ports are constituted by public member variables of type `sca_tdf::sca_in<T>` and `sca_tdf::sca_out<T>`, respectively. With the template argument T of the ports, the data type of the signal is specified. The ports can be only bound to an `sca_tdf::sca_signal<T>` channel of matching data type T . Each TDF signal requires to be bound to one driving output port and zero to n reading input ports. Like in SystemC, a constructor needs to be defined for each TDF module, which has always as first argument the module name of type `sc_core::sc_module_name`. It is good practice to initialise the base name of each of its ports in the constructor to facilitate later debugging. The `processing()` member function of `sca_tdf::sca_module` needs to be always overloaded to define the behaviour. In the case of the mixer, the input value read from each of the two input ports are multiplied with each other and the result is written to the output port.

Listing 5.1: TDF model of the mixer.

```

1  class mixer : public sca_tdf::sca_module {
2  public:
3      sca_tdf::sca_in<double> in1, in2; // Inputs.
4      sca_tdf::sca_out<double> out;    // Output.
5
6      // Construct mixer and name its ports.
7      mixer(sc_core::sc_module_name nm)
8      : in1("in1"), in2("in2"), out("out")
9      {}
10
11 protected:
12     // Multiply input samples and write result to the out port.
13     void processing() {
14         out.write(in1.read() * in2.read());
15     }
16 };

```

The low-pass filter block is also modelled as a single-rate TDF module. Its implementation shown in Listing 5.2 is slightly more complicated, as the filter's cut-off frequency f_c and the gain H_0 can be parameterised by passing them as arguments to the module's constructor. The filter behaviour is implemented using an embedded linear dynamic equation described by a Laplace Transfer Function (LTF) in numerator-denominator form. TDF modules may also embed LTFs in zero-pole form or state-space equations. The coefficients of the LTF's numerator `num_` and denominator `den_` are stored as member variables and are initialised in the module's constructor using f_c . The filter gain is also stored in a member variable (`H0_`). The solution of the LTF is handled by an object of type `sca_tdf::sca_ltf_nd`, which has to be created as a member variable of the module, as it needs to keep track of its internal state between the calls to the processing member function, in which the newly read input sample is handed over to it and the filtered result is calculated.

The ADC is modelled as a multi-rate TDF module. Its implementation shown in Listing 5.3 is a bit more complex due to its various parameters and the support for multi-rate. It is implemented as a template class so that the size `NBITS` of the output word, which will be read by the DSP, can be configured at module instantiation. The output word is written to a TDF converter port of type `sca_tdf::sca_de::sca_out<T>`. This port can be bound to a regular `sc_core::sc_signal<T>` to interface a TDF module with a regular SystemC module. Likewise, a TDF converter port of type `sca_tdf::sca_de::sca_in<T>` is available to read values from connected SystemC modules. This shows the general concept of synchronisation in the SystemC AMS extensions: Each MoC implements dedicated converter ports or modules as communication interfaces to the other continuous-time MoCs and the DE MoC of the SystemC kernel. The input range $in_{\min} \leq in \leq in_{\max}$, which can be quantised by the ADC is specified by the constructor parameters `inmin` and `inmax`. The remaining parameters of the ADC are related to the configuration of its multi-rate behaviour, which has to be done for each TDF module in its `set_attributes()` member function, which is called by the TDF MoC during elaboration to configure the TDF parameters of the cluster. In this callback, each module can specify:

- The time step of the activation of its `processing()` member function using the `set_timestep()` member function of the module.
- The time step between individual samples read from a port using the `set_timestep()` member

Listing 5.2: TDF model of the low-pass filter using a Laplace transfer function.

```

1  class lp_filter_ltf : public sca_tdf::sca_module {
2  public:
3      sca_tdf::sca_in<double> in;    // Input.
4      sca_tdf::sca_out<double> out; // Output.
5
6      // Construct filter having the specified cut-off frequency fc and gain H0.
7      lp_filter_ltf(sc_core::sc_module_name nm, double fc, double H0 = 1.0)
8      : in("in"), out("out"), H0_(H0), ltf_(), num_(), den_()
9      {
10         // Initialise LTF numerator and denominator.
11         num_(0) = 1.0;
12         den_(0) = 1.0; den_(1) = 1.0/(2.0 * M_PI * fc);
13     }
14
15 protected:
16     // Apply LTF to read input samples and write result to out port.
17     void processing() {
18         out.write(ltf_(num_, den_, in.read(), H0_));
19     }
20 private:
21     double H0_; // Filter gain.
22     sca_tdf::sca_ltf_nd ltf_; // Laplace transfer function object.
23     sca_util::sca_vector<double> num_, den_; // Numerator and denominator of LTF.
24 };

```

function of the port.

- How many samples it wants to read/write from/to a port per activation of its `processing()` member function using the `set_rate()` member function of the port.
- By how many discrete port time steps a read/written sample will be delayed using the `set_delay()` member function of the port.

By default, the rate of a port is one and its delay is zero. Therefore, the `set_attributes()` member function did not need to be implemented for the mixer and low-pass filter modules. At least one time step needs to be assigned per TDF cluster. It will be propagated throughout the cluster taking the set port rates into account. Figure 5.2 illustrates this process for our RF front end model. The source of the time step propagation is in this case the ADC, which specifies its module time step (T_m) based on the sampling period T_s (T_s) of the baseband signal written to the DSP block. Of course, the RF signal needs to be sampled at a much higher rate than the base band signal. This ratio is configured through the parameter R_{RF} (`Rrf`) that is used to specify the rate (R) of the ADC's input port. The resulting port time step (T_p) is $\frac{T_s}{R_{RF}}$. As the mixer and filter operate at single-rate, their port and module time steps will be after time step propagation also equal to $\frac{T_s}{R_{RF}}$. The parameter φ_{RF} (`phirf`) specifies which sample out of R_{RF} samples of the down-converted and filtered RF input signal is quantised by the ADC. To model the time that the ADC needs for the quantisation, a delay of D_q sampling periods is set on its output port. The initial samples of delayed ports must be initialised before the simulation starts. This has to be done in a separate callback of the TDF module, which is called `initialize()`, so that the TDF MoC can finish first its port elaboration after the `set_attributes()` callbacks of all TDF modules have been executed. The ADC model initialises its output samples to zero.

Listing 5.3: TDF model of the ADC.

```

1  template<int NBITS>
2  class ad_converter : public sca_tdf::sca_module {
3  public:
4      sca_tdf::sca_in<double> in;
5      sca_tdf::sca_de::sca_out<sc_dt::sc_uint<NBITS> > out; // Converter port TDF->DE.
6
7      ad_converter(sc_core::sc_module_name nm, sca_core::sca_time Ts,
8                  double inmin, double inmax,
9                  unsigned long Dq, unsigned long Rrf, unsigned long phirf)
10     : in("in"), out("out"), Ts_(Ts), inmin_(inmin), inmax_(inmax),
11       Dq_(Dq), Rrf_(Rrf), phirf_(phirf)
12     {
13         sc_assert(phirf < Rrf); // Check parameters for consistency.
14         sc_assert(inmin < inmax);
15     }
16
17 protected:
18     // Specify the module and port attributes.
19     void set_attributes() {
20         this->set_timestep(Ts_); // Module time step.
21         in.set_rate(Rrf_); // Input rate.
22         out.set_delay(Dq_); // Output delay.
23     }
24
25     // Initialise delayed output samples.
26     void initialize() {
27         for (unsigned long i = 0; i < out.get_rate(); ++i) {
28             out.initialize(0);
29         }
30     }
31
32     void processing() {
33         double val = in.read(phirf_); // Read RF sample at index phirf_.
34         // Quantise read sample and write it to the delayed output port.
35         if (val < inmin_) {
36             out.write(0);
37         } else if (val > inmax_) {
38             out.write((1 << NBITS) - 1);
39         } else {
40             sc_dt::sc_uint<NBITS>
41             q_val = ((val - inmin_) / (inmax_ - inmin_) * ((1 << NBITS) - 1));
42             out.write(q_val);
43         }
44     }
45
46 private:
47     sca_core::sca_time Ts_; // Output sample period of ADC.
48     double inmin_, inmax_; // Input value range to be quantised.
49     unsigned long Dq_; // Output delay due to quantisation process.
50     unsigned long Rrf_; // Input rate to oversample RF signal.
51     unsigned long phirf_; // Index to input sample to be quantised.
52 };

```

Listing 5.4: Structural model of the RF front end.

```

1  template<int NBITS = 8>
2  class rf_front_end : public sc_core::sc_module {
3  public:
4      // Ports.
5      sca_tdf::sca_in<double> rf_in, osc_in;
6      sc_core::sc_out<sc_dt::sc_uint<NBITS> > dig_out;
7
8      // Internal components.
9      mixer mxr_1;
10     lp_filter_lsf lpf_1;
11     ad_converter<NBITS> adc_1;
12
13     // Internal signals.
14     sca_tdf::sca_signal<double> mxr_sig, lp_sig;
15
16     // Construct RF front end using the passed parameters for the components.
17     rf_front_end(sc_core::sc_module_name nm,
18                 sca_core::sca_time Ts, double inmin, double inmax,
19                 unsigned long Dq = NBITS/2, unsigned long Rrf = 1000000,
20                 unsigned long phirf = 0, double fc = 200.0e3, double H0 = 1.0)
21     : mxr_1("mxr_1"), lpf_1("lpf_1", fc, H0),
22       adc_1("adc_1", Ts, inmin, inmax, Dq, Rrf, phirf)
23     {
24         // Specify connectivity.
25         mxr_1.in1(rf_in);
26         mxr_1.in2(osc_in);
27         mxr_1.out(mxr_sig);
28
29         lpf_1.in(mxr_sig);
30         lpf_1.out(lp_sig);
31
32         adc_1.in(lp_sig);
33         adc_1.out(dig_out);
34     }
35 };

```

The structural assembly of these three TDF modules to the RF front end model has to be done in a regular SystemC module that inherits from `sc_core::sc_module` and *not* from `sca_tdf::sca_module`. The RF front end module (Listing 5.4) specifies its ports, internal signals, and internal modules as member variables. These are initialised in the initialisation list of the constructor, which passes the relevant parameters from its arguments to the constructor of the individual member variables. Finally, it specifies the connectivity by binding the ports of its internal modules to the defined signals or ports.

To illustrate the possible interaction between SystemC's discrete-event simulation kernel and the TDF MoC, let us consider a modification of the low-pass filter model as shown in Listing 5.5. The `ctrl` port allows the control of the output amplification. It is intended to be bound to an `sc_core::sc_signal<bool>` channel. The value of this port is sampled at the fixed time steps used to activate the TDF module.

The TDF MoC supports further architecture level exploration through, e.g., the addition of non-ideal behaviours (e.g., noise and distortion in the mixer), the refinement of data representation (e.g., different

Listing 5.5: Event-controlled low-pass filter model with switchable gain.

```

1  class lp_filter_ltf_ctrl : public sca_tdf::sca_module {
2  public:
3      sca_tdf::sca_in<double> in;           // Input.
4      sca_tdf::sca_de::sca_in<bool> ctrl;   // Gain control input.
5      sca_tdf::sca_out<double> out;        // Output.
6
7      // Construct filter having the specified cut-off frequency fc and gains H0, H1.
8      lp_filter_ltf_ctrl(sc_core::sc_module_name nm, double fc,
9                          double H0 = 1.0, double H1 = 2.0)
10     : in("in"), ctrl("ctrl"), out("out"), H0_(H0), H1_(H1), ltf_(), num_(), den_()
11     { /* ... */ }
12
13 protected:
14     // Apply LTF to read input samples and write result to out port.
15     void processing() {
16         double h = ctrl.read() ? H1_ : H0_; // Set gain according to control signal.
17         out.write(ltf_(num_, den_, in.read(), h));
18     }
19 private:
20     double H0_, H1_;                       // Filter gains.
21     sca_tdf::sca_ltf_nd ltf_;              // Laplace transfer function object.
22     sca_util::sca_vector<double> num_, den_; // Numerator and denominator of LTF.
23 };

```

bit widths) or structural refinement. Signal processing applications can also benefit from the TDF MoC by exploring the impacts of more specific parameters such as delays or data rates. The key to such systematic architectural studies are parameterisable models as they have been presented in this section.

5.2.2. Structural refinement using the Linear Signal Flow and Electrical Linear Network Models of Computation

The SystemC AMS extensions also support the Linear Signal Flow (LSF) and Electrical Linear Network (ELN) MoCs. As described in Section 5.2, the first allows the description of non-conservative linear dynamic behaviour and the latter the description of conservative linear behaviour. Both MoCs are similar in their usage, as they only allow the structural description of an LSF/ELN model using predefined primitive LSF/ELN modules. To this end, the SystemC AMS extensions offer a fixed set of typical signal flow modules (e.g., adder, gain, and integrator blocks) and electrical primitives (e.g., resistor, inductor, capacitor, and (controlled) voltage/current sources). These primitives can be parameterised and interconnected to macro models in a regular SystemC module using in the case of the LSF MoC signals (of type `sca_lsf::sca_signal`) bound to input and output ports (`sca_lsf::sca_in` and `sca_lsf::sca_out`, respectively) and in the case of the ELN MoC using nodes and reference nodes (`sca_eln::sca_node` and `sca_eln::sca_node_ref`, respectively) bound to terminals (`sca_eln::sca_terminal`). Both, the LSF MoC and ELN MoC, provide converter modules to interface with the TDF MoC and DE MoC. Thus, the SystemC AMS extensions do not only support the structural refinement within one MoC, but also across different MoCs. As an example, let us refine the low-pass filter model from Listing 5.2. Figure 5.3 shows the schematic of the equivalent LSF model. Listing 5.6 shows its implementation as a SystemC module containing the structural de-

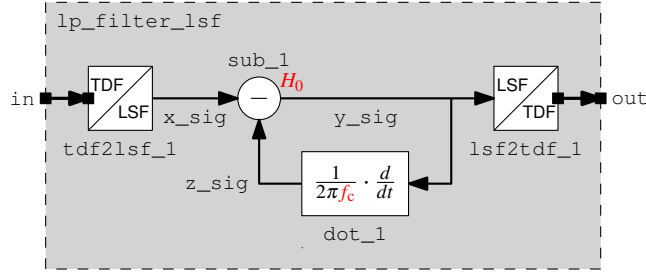


Figure 5.3.: LSF model of a first-order low-pass filter with LSF converter modules to connect the module via its TDF ports to other TDF modules.

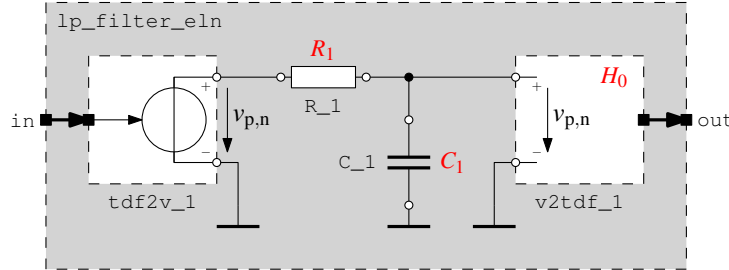


Figure 5.4.: ELN model of a first-order low-pass filter with ELN converter modules to connect the module via its TDF ports to other TDF modules.

scription. Due to the instantiation of the appropriate converter modules, its port interface is identical to the one of the TDF module `lp_filter_ltf`. Also their module parameters are identical, so that they can be interchanged in the RF front end module (Listing 5.4) by just changing the type of the member variable `lpf_1` to `lp_filter_lsf`. Similarly, an equivalent ELN model of the low-pass filter can be developed (Figure 5.4, Listing 5.7). Again, the port interface stayed compatible to `lp_filter_ltf` through the insertion of appropriate converter modules. However, the constructor now proposes R_1 and C_1 as parameters instead of f_c , which requires a small modification in the parameterisation of the `lpf_1` module when swapping the TDF module with the ELN module in the RF front end module.

5.2.3. Conclusions about the OSCI SystemC AMS extensions

The Timed Data Flow (TDF) MoC, as defined in the OSCI SystemC AMS extensions, provides a high level of abstraction for modelling continuous-time behaviours using discrete-time semantics and mixing them with discrete-event models of hardware and even untimed models of software (e.g., using mixed Transaction Level/AMS models). If required, refined models can be created using the Linear Signal Flow (LSF) and Electrical Linear Network (ELN) MoCs, which use continuous-time semantics and which are also defined as part of the SystemC AMS extensions. However, it has to be clearly stated that their flexibility is reduced compared to TDF models, as they only allow the creation of macro models using interconnected predefined primitive LSF/ELN modules with purely linear behaviour. This is the sacrifice to be made to obtain satisfying simulation performances at the system level. To some degree, this limitation can be leveraged by embedding the LSF/ELN models into TDF models or discrete-event SystemC models, which control them via one of the several proposed primitives with DE/TDF control inputs for their component parameter. More detailed information about the usage of the

Listing 5.6: LSF model of the lowpass filter.

```

1 class lp_filter_lsf : public sc_core::sc_module {
2 public:
3     sca_tdf::sca_in<double> in;    // Input.
4     sca_tdf::sca_out<double> out;  // Output.
5
6     sca_lsf::sca_signal x_sig, y_sig, z_sig; // Internal LSF signals.
7
8     // Internal LSF primitives.
9     sca_lsf::sca_tdf::sca_source tdf2lsf_1; // Converter TDF -> LSF input.
10    sca_lsf::sca_sub sub_1;                // Subtractor.
11    sca_lsf::sca_dot dot_1;                // Differentiator.
12    sca_lsf::sca_tdf::sca_sink lsf2tdf_1;   // Converter LSF -> TDF output.
13
14    // Construct filter having the specified cut-off frequency and gain.
15    lp_filter_lsf(sc_core::sc_module_name nm, double fc, double H0 = 1.0)
16    : in("in"), out("out"), x_sig("x_sig"), y_sig("y_sig"), z_sig("z_sig"),
17      tdf2lsf_1("tdf2lsf_1"), sub_1("sub_1", H0),
18      dot_1("dot_1", 1.0 / (2.0 * M_PI * fc)), lsf2tdf_1("lsf2tdf_1")
19    {
20        // Specify connectivity.
21        tdf2lsf_1.inp(in); tdf2lsf_1.y(x_sig);
22
23        sub_1.x1(x_sig); sub_1.x2(z_sig);
24        sub_1.y(y_sig);
25
26        dot_1.x(y_sig); dot_1.y(z_sig);
27
28        lsf2tdf_1.x(y_sig); lsf2tdf_1.outp(out);
29    }
30 };

```

Listing 5.7: ELN model of the lowpass filter.

```

1 class lp_filter_eln : public sc_core::sc_module {
2 public:
3   sca_tdf::sca_in<double> in;    // Input.
4   sca_tdf::sca_out<double> out;  // Output.
5
6   sca_eln::sca_node n1, n2;      // Electrical nodes.
7   sca_eln::sca_node_ref gnd;     // Electrical reference node.
8
9   // Internal ELN primitives.
10  sca_eln::sca_tdf::sca_vsource tdf2v_1; // Converter TDF -> voltage.
11  sca_eln::sca_r R_1;              // Resistor.
12  sca_eln::sca_c C_1;              // Capacitor.
13  sca_eln::sca_tdf::sca_vsink v2tdf_1;  // Converter voltage -> TDF.
14
15  // Construct filter having the specified cut-off frequency and gain.
16  lp_filter_eln(sc_core::sc_module_name nm, double R1, double C1, double H0 = 1.0)
17  : in("in"), out("out"), n1("n1"), n2("n2"), gnd("gnd"),
18    tdf2v_1("tdf2v_1"), R_1("R_1", R1), C_1("C_1", C1), v2tdf_1("v2tdf_1", H0)
19  {
20    // Specify connectivity.
21    tdf2v_1.inp(in);
22    tdf2v_1.p(n1); tdf2v_1.n(gnd);
23
24    R_1.p(n1); R_1.n(n2);
25
26    C_1.p(n2); C_1.n(gnd);
27
28    v2tdf_1.p(n2); v2tdf_1.n(gnd);
29    v2tdf_1.outp(out);
30  }
31 };

```

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

SystemC AMS extensions can be found in the user's guide [14] accompanying its LRM [131].

The demonstrated RF front end example shows that the SystemC AMS extensions are in their first standardised version well-suited for the design of electrical AMS SoCs, especially for communication systems [46] and for DSP applications. Their limitations concerning the description of digitally assisted analogue and multiphysical components with nonlinear behaviour are recognised. Therefore, efforts are underway, e.g., to integrate a nonlinear network MoC into the AMS extensions with capabilities similar to VHDL-AMS or Verilog-AMS [168]. However, as the complexity of the resulting DAE system will not differ from an equivalent VHDL-AMS or Verilog-AMS model, no considerable gains in terms of simulation performance can be expected, as the underlying solver algorithms are expected to be the same.

The simulation of nonlinear conservative models is just one side of the problem. The other is the definition of a user-friendly syntax for the authoring of these nonlinear models, which is constrained by the syntax of the C++ language. Furthermore, the verification of the correct assembly of such heterogeneous models and of the coherency of the implemented equations gets even more important for multiphysical systems. Therefore, this Ph.D. thesis work elaborated another way to describe *energy-conserving multiphysical system* components with *nonlinear behaviour* in a *formal* and *consistent* way at a *higher level of abstraction*. To this end, the next section reviews the modelling of multiphysical systems on different levels of abstraction.

5.3. Modelling Multiphysical Systems on Different Abstraction Levels

In this section, it will be shown how to derive a SystemC AMS extensions compatible model of a multiphysical system by successively rising the abstraction level without losing the link to the physical domain. We will use an electromechanical transducer (electrostatic comb-drive actuator) linked to a micromechanical resonator as an example (Figure 5.5).

5.3.1. Using Domain-Specific Modelling Primitives

In a first step, the system can be modelled using *domain-specific primitives*, in our case (Figure 5.5a): voltage source, resistor for the electrical domain and mass, spring, damper for the mechanical domain. The comb drive actuator is part of both domains and acts electrically as a capacitor, which capacitance $C_{\text{trans}}(x)$ depends on the current displacement x , and mechanically as a force source $F_{\text{trans}}(q, x)$, which value depends on the electrical charge q , stored on the capacitor, and on the displacement x . The resulting model accurately represents the physical structure of the system. However, this approach requires a simulator to provide model implementations for each primitive of each supported physical domain. This implies an overhead as there exist analogies between the primitives of different domains (e.g., resistor/damper, capacitor/spring, inductor/mass). This has been exploited regularly to, e.g., simulate mechanical resonators using an electrical simulator such as SPICE. However, this approach sacrifices clarity, which easily leads to modelling mistakes. Furthermore, the SystemC AMS extensions currently only offer (linear) electrical primitives on this abstraction level.

5.3.2. Using Generic Bond Graph Primitives

The *bond graph formalism* [94] is taking advantage of these analogies to unify the description of multiphysical systems through a reduced set of *generic primitives*. The domain-specific description (e.g., electrical circuit, mechanical multibody system, rigid bodies, fluidic networks, thermal networks) is mapped in a systematic way on a graph (Figures 5.5b and 5.5c) describing the energy flow through the

5.3. Modelling Multiphysical Systems on Different Abstraction Levels

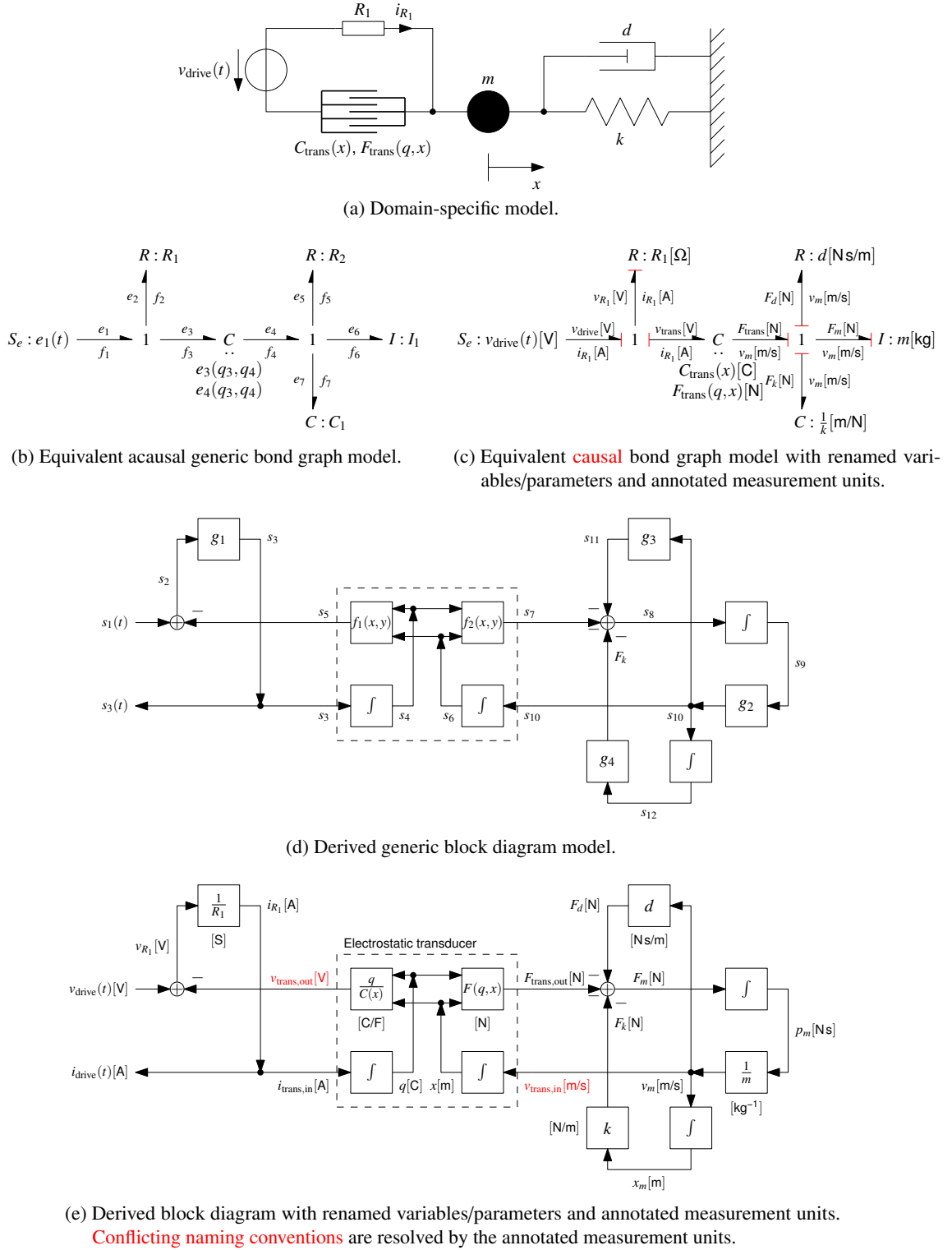


Figure 5.5.: Equivalent models of an electromechanical transducer linked to a micromechanical resonator.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

multiphysical system. The energy link between the ports of two primitives Pr_i and Pr_k is represented with an half-arrow-shaped *bond*: $Pr_i \xrightarrow[e]{e} Pr_k$. Associated to each bond are an *effort* e and a *flow* f variable. They are called *power variables* because their product is the power P :

$$P(t) = e(t) \cdot f(t) \quad (5.1)$$

By definition, the half arrow points into the direction, in which the power flows for positive e and f .

For the description of dynamic systems, two other variable types are important, which belong to the class of *energy variables*: The (*generalised*) *momentum* $p(t)$ is defined as the time integral of an effort:

$$p(t) = \int e(t') dt' = p_0 + \int_{t_0}^t e(t') dt' \quad (5.2)$$

The (*generalised*) *displacement* $q(t)$ is defined as the time integral of a flow:

$$q(t) = \int f(t') dt' = q_0 + \int_{t_0}^t f(t') dt' \quad (5.3)$$

with p_0 and q_0 the initial values of p and q , respectively, at the time t_0 . Both equations can also be written in differential form:

$$\frac{dp(t)}{dt} = e(t) \quad dp = e dt \quad (5.2a)$$

$$\frac{dq(t)}{dt} = f(t) \quad dq = f dt \quad (5.3a)$$

The energy $E(t)$ is defined as the time integral of the power $P(t)$:

$$E(t) = \int P(t') dt' = \int e(t') f(t') dt' \quad (5.4)$$

The energy can be also expressed in terms of the energy variables by inserting (5.2a) and (5.3a) in (5.4):

$$E(t) = \int e(t') dq(t') = \int f(t') dp(t') \quad (5.5)$$

$$E(q) = \int e(q') dq' \quad (5.5a)$$

$$E(p) = \int f(p') dp' \quad (5.5b)$$

The relation between the power and energy variables can be visualised in the so-called *tetrahedron of state* shown in Figure 5.6. The definition for the *generalised* power and energy variables is independent of a particular physical domain. Table 5.1 summarises their meaning in the context of different domains.

The energy exchange through a bond can be also interpreted as a bilateral signal flow, since the subsystems at each end act as each others load and thus cause the effort and flow variables to act in opposite directions. This can be used to determine the computational direction, which is indicated by a perpendicular stroke at one end of the bond. This *causal stroke* states that at this side the effort variable e is known (it acts as an input) and f can be calculated as a function $f := \Phi_k^{-1}(e)$. Consequently, the flow f is known on the other side of the bond and acts as an input to a function to calculate the effort $e := \Phi_i(f)$. Figure 5.7 illustrates this relation. The equations describing the primitive's behaviour

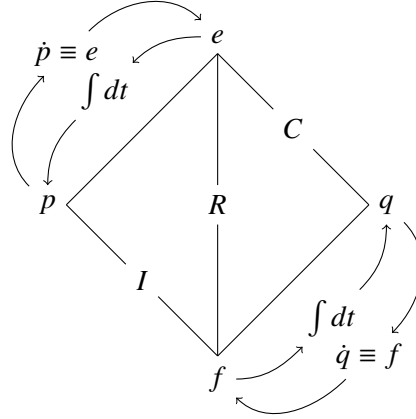


Figure 5.6.: The tetrahedron of state visualises how the power and energy variables are related through the three basic 1-port primitives R , C , and I [adapted from 94].

Table 5.1.: Power and energy variables for different physical domains.

Physical domain	Effort $e(t)$	Flow $f(t)$	(Generalised) momentum $p(t)$	(Generalised) displacement $q(t)$
Electrical	Voltage, [v] = V	Current, [i] = A	Flux linkage variable, [λ] = V s	Charge, [q] = A s
Mechanical translational	Force, [F] = N	Velocity, [v] = m/s	Momentum, [p] = N s	Displacement, [x] = m
Mechanical rotational	Torque, [τ] = N m	Angular velocity, [ω] = rad/s	Angular momentum, [p_τ] = N m s	Angle, [θ] = rad
Hydraulic	Pressure, [p] = Pa	Volume flow rate, [Q] = m ³ /s	Pressure momentum, [p_p] = N s/m ²	Volume, [V] = m ³

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

impose a *required*, *preferred* (e.g., due to numerical reasons), or *free* causality (*effort-in* or *effort-out*) on the primitive's ports. The causalities assigned to one port need to be propagated as constraints to all related ports. Methods like the Sequential Causality Assignment Procedure (SCAP) [94] exist to systematically complete the causality of a bond graph, which is one of their main advantages together with the compact visualisation of the energy flow and of the computational structure. The assigned causalities allow to sort the primitives' equations in the right order for model execution and to do some further formal checks on the model: the number of states and non-states in the system, the presence of algebraic loops during model execution, or if it is an ill-posed model.

Three generalised 1-port primitives represent the resistive R , inertial I , and capacitive C behaviour independent of the considered physical domain. Energy sources are modelled as effort source S_e and flow source S_f primitives. Quantity transformations (also across domain boundaries) are represented through the transformer TF and gyrator GY 2-port primitives, which ensure the conservation of energy. Multi-port junction primitives represent explicitly the interconnection of primitives, which are exposed to a common effort (0-junction) or a common flow (1-junction). All primitives can have nonlinear characteristic equations. Transformer, gyrator, and sources can be modulated by an external signal. Table 5.2 summarises the basic primitives of a bond graph with the permitted causality assignments and their defining equations. Please note that the bond direction is not imposed by the bond graph primitives. However, their parameter definition assumes the indicated bond directions. Additionally, special multiport versions of R , I , C primitives can be used to describe distributed component behaviours of, e.g., a beam, fields, or in our case the electromechanical transducer (Figure 5.5c). Due to the generic names for the primitives, variables, and parameters and partially overlapping standards for quantity symbols (e.g., v for voltage and velocity), the link to the physical domain is not anymore forcibly kept through their name, but rather through the quantity type (measurement unit) associated to the variables and parameters (Figures 5.5b and 5.5c). Therefore, extra precaution has to be taken when dealing with variable and parameter names and their associated quantity type. For this reason support for dimensional analysis has been developed for the SystemC AMS extensions, as presented in Section 5.4.

Complex systems such as heterogeneous AMS SoCs require a hierarchical description of their structure. This can be achieved by using the more abstract *word bond graphs* [94], which add the usage of “macros” to the classic bond graph modelling to represent a multiport component of the overall system. Such systems are also often characterised through the presence of digital units, which control the continuous-time parts through feedback and switching of energy flows due to digital signals. The switching can be modelled using idealised controlled junctions (Table 5.3). Their presence indicate a *hybrid bond graph* [16]. An active junction constrains the causality of attached bonds like a normal junction. When it gets inactive, it imposes a zero on the power variable, which is common to all attached bonds, leading to a causality change. In electrical terms this means that a parallel connection modelled by a 0-junction is “short-circuited” and a series connection modelled by a 1-junction is “left open”. The causality change needs to be propagated into the graph and requires a solver reinitialisation imposing a simulation performance penalty.

The standardised SystemC AMS extensions [131] do not support the bond graph formalism. Adding this support in form of a new MoC has been a principal goal of this Ph.D. thesis work. The results are presented in Section 5.7.

5.3.3. Using a Block Diagram

The causality assignment allows for a natural integration of bond graphs with block diagrams and their transformation in the latter (Figures 5.5c to 5.5e). The *block diagram formalism* does not guarantee

5.3. Modelling Multiphysical Systems on Different Abstraction Levels

Table 5.2.: Basic primitives of a bond graph with their legal causality assignments.

Name	Symbol	Defining relation		Examples from electrical, mechanical, and hydraulic domains
		General case	Linear case	
Effort source	$S_e \xrightarrow[e]{e}$	$e(t)$ given, $f(t)$ arbitrary		<ul style="list-style-type: none"> Voltage, force, pressure sources
Modulated effort source	$\xrightarrow[e(t)]{e} S_e \xrightarrow[e]{e}$	$e(t)$ given through signal, $f(t)$ arbitrary		
Flow source	$S_f \xrightarrow[f]{e}$	$f(t)$ given, $e(t)$ arbitrary		<ul style="list-style-type: none"> Current, velocity, volume flow rate sources
Modulated flow source	$\xrightarrow[f(t)]{f} S_f \xrightarrow[f]{e}$	$f(t)$ given through signal, $e(t)$ arbitrary		
(Generalised) resistor	$\xrightarrow[f]{e} R$	$e = \Phi_R(f)$	$e = Rf$	<ul style="list-style-type: none"> Electrical resistor Translational/rotational damper Hydraulic throttle
	$\xrightarrow[e]{f} R$	$f = \Phi_R^{-1}(e)$	$f = \frac{1}{R}e$	
(Generalised) capacitor	$\xrightarrow[f=\dot{q}]{e} C$	$q = \Phi_C(e)$	$q = Ce$	<ul style="list-style-type: none"> Electrical capacitor Spring, torsion bar Gravity tank
	$\xrightarrow[f=\dot{q}]{e} C$	$e = \Phi_C^{-1}(q)$	$e = \frac{1}{C}q$	
(Generalised) inertia	$\xrightarrow[f]{e=\dot{p}} I$	$p = \Phi_I(f)$	$p = If$	<ul style="list-style-type: none"> Electric inductor Mass, rotating disk Fluid-filled pipe section
	$\xrightarrow[f]{e=\dot{p}} I$	$f = \Phi_I^{-1}(p)$	$f = \frac{1}{I}p$	
(Generalised) transformer	$\xrightarrow[f_1]{e_1} TF \xrightarrow[f_2]{e_2}$	$e_1 = me_2, f_2 = mf_1$		<ul style="list-style-type: none"> Electrical transformer Ideal rigid lever, gear pair Hydraulic ram
	$\xrightarrow[f_1]{e_1} TF \xrightarrow[f_2]{e_2}$	$e_2 = \frac{1}{m}e_1, f_1 = \frac{1}{m}f_2$		
Modulated (generalised) transformer	$\xrightarrow[f_1]{e_1} \xrightarrow[m(t)]{MTF} \xrightarrow[f_2]{e_2}$	$e_1 = m(t)e_2, f_2 = m(t)f_1$		<ul style="list-style-type: none"> Autotransformer with wiper Geometric transformations
(Generalised) gyrator	$\xrightarrow[f_1]{e_1} GY \xrightarrow[f_2]{e_2}$	$e_1 = rf_2, e_2 = rf_1$		<ul style="list-style-type: none"> Electrical gyrator Gyroscope Voice coil transducer
	$\xrightarrow[f_1]{e_1} GY \xrightarrow[f_2]{e_2}$	$f_2 = \frac{1}{r}e_1, f_1 = \frac{1}{r}e_2$		
Modulated (generalised) gyrator	$\xrightarrow[f_1]{e_1} \xrightarrow[r(t)]{MGY} \xrightarrow[f_2]{e_2}$	$e_1 = r(t)f_2, e_2 = r(t)f_1$		<ul style="list-style-type: none"> Gyroscope with variable rotor speed Voice coil with variable transduction coefficient
Flow junction, 0-junction, common effort junction	$\begin{array}{c} e_n \downarrow f_n \\ \xrightarrow[f_1]{e_1} 0 \xrightarrow[f_j]{e_j} \\ \uparrow f_i \\ e_i \end{array}$	$e_1 = \dots = e_i = \dots = e_n$ $f_i = -\left(\sum_{j=1, j \neq i}^n f_j\right)$		<ul style="list-style-type: none"> Electrical parallel connexion Situation with a single force and n velocities summing up to zero Hydraulic parallel connexion
Effort junction, 1-junction, common flow junction	$\begin{array}{c} e_n \downarrow f_n \\ \xrightarrow[f_1]{e_1} 1 \xrightarrow[f_j]{e_j} \\ \uparrow f_i \\ e_i \end{array}$	$f_1 = \dots = f_i = \dots = f_n$ $e_i = -\left(\sum_{j=1, j \neq i}^n e_j\right)$		<ul style="list-style-type: none"> Electrical series connexion Dynamic equilibrium of n forces associated with a single velocity Hydraulic series connexion

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

$$\begin{aligned}
 Pr_i \xrightarrow[e]{e} Pr_k &\Rightarrow \boxed{Pr_i \xleftarrow[f]{e} Pr_k} \Rightarrow \begin{aligned} f &:= \Phi_k^{-1}(e) \\ e &:= \Phi_i(f) \end{aligned} \\
 Pr_i \xrightarrow[f]{e} Pr_k &\Rightarrow \boxed{Pr_i \xleftarrow[e]{f} Pr_k} \Rightarrow \begin{aligned} f &:= \Phi_i^{-1}(e) \\ e &:= \Phi_k(f) \end{aligned}
 \end{aligned}$$

Figure 5.7.: Interpretation of a bond as a bilateral signal flow.

Table 5.3.: Controlled junctions in hybrid bond graphs for modelling discrete switching of energy flows.

Name	Symbol	On-state	Off-state
Controlled 0-junction			
Controlled 1-junction			

itself the conservation of energy in the system. It is emulated by the way the block diagram primitives are interconnected. The block diagram primitives (scaler, integrator, summer, etc.) are so generic that they do not establish a direct link to the modelled physical effect (Figure 5.5d). That's why it becomes paramount to annotate each signal and parameter with its quantity type to reestablish the link and to be able to check the system model on the structural and equation level for consistency (Figure 5.5e). Otherwise, partially overlapping symbol names for different kinds of quantities (e.g., in Figure 5.5e $v_{\text{trans,in}}$ for the velocity input and $v_{\text{trans,out}}$ for the voltage output of the transducer sub-model) could result in hard to spot interconnection mistakes during the model creation by the designer.

The close relationship between causal bond graphs and block diagrams also allows to mix them in a single model. An example is given in Figure 5.8 in form of a car wheel model of an electronically controlled suspension system incorporating a semi-active damper and a fast load-leveler [94]. Both ways of representing the systems are given: the “classic” domain-specific way and the way using bond graphs for the energy conservation part and block diagrams for the signal processing part. The blocks in the signal flow graph can take the calculated power or energy variables as input and can modulate the sources or element parameters of the bond graph.

Block diagram models of physical components are in general hard to reuse as all physical quantities have already been assigned their input/output roles till the interface of the component. This seriously limits the interconnection options with other physical components (e.g., only series connection or only parallel connection). However, they are entirely causal, which allows for a procedural and thus very efficient model execution, e.g., with the TDF MoC of the SystemC AMS extensions (Section 5.2.1).

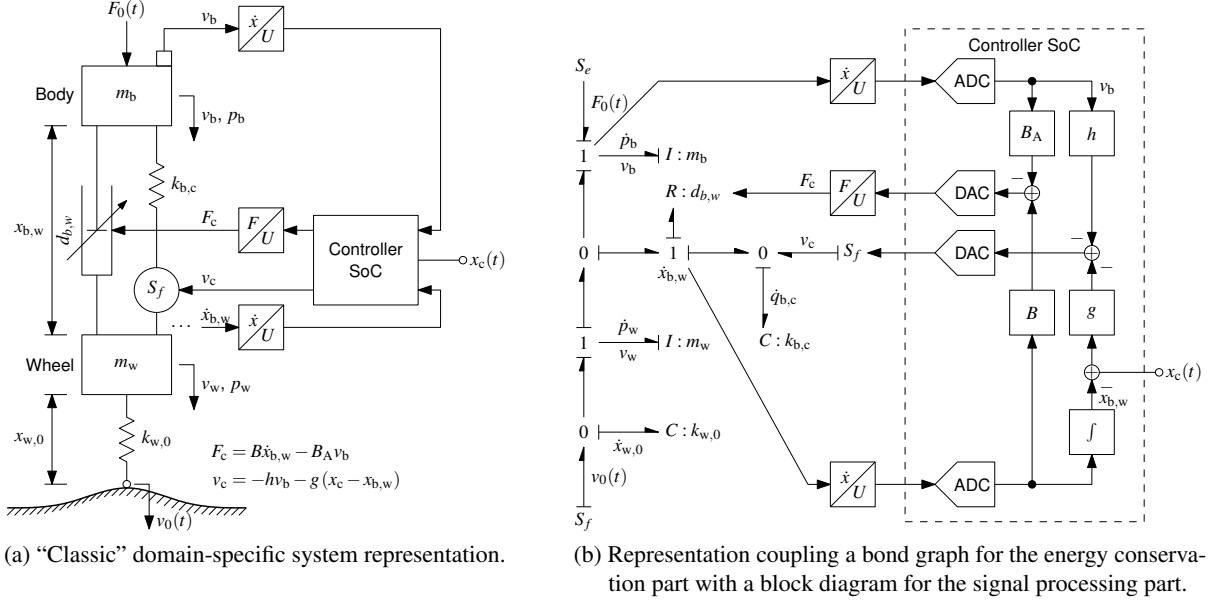


Figure 5.8.: Car wheel model of an electronically controlled suspension system incorporating a semi-active damper and a fast load-leveller [94].

5.4. Integrating Dimensional Analysis with SystemC AMS extensions

Sections 5.1 and 5.3 showed the importance of annotating parameters and variables in multiphysical system models with their measurement unit as part of their *quantity data type* to not lose the link to physical domain when using more abstract modelling formalisms like bond graphs and block diagrams. It was also shown that current HDLs insufficiently support the concept of quantity data types. Therefore, automated checks of the model assembly and equation coherency using dimensional analysis are often not available in the simulation tools supporting these HDLs. The openness of the SystemC AMS simulation framework and the powerful C++ programming language makes the definition of quantity data types supporting dimensional analysis and their integration with the SystemC AMS extensions possible.

The integration of dimensional analysis with the SystemC AMS extensions requires several steps:

1. Implement a quantity data type with unit annotation and dimensional analysis in C++ without prohibitive runtime penalty.
2. Facilitate the debugging of compiler errors related to the quantity data type.
3. Facilitate the consistent usage of this quantity data type in SystemC models of multiphysical systems by offering a library that helps to reduce code overhead due to the reimplementations of very similar behaviours for different quantity types.

In this section, the focus will be on the first two steps, as they lay the foundation for using quantity types and dimensional analysis in SystemC models. Facilitating the usage of the quantity types will be addressed in separate sections for the TDF MoC (Section 5.6) of the standard SystemC AMS extensions and for the new Bond Graph (BG) MoC (Section 5.7) proposed by this Ph.D. thesis work.

5.4.1. Compile-time dimensional analysis with Boost.Units

For the first step, a mature implementation is available in form of the peer-reviewed *Boost.Units* library [158], which has been used in this work. It implements dimensional analysis for arbitrary systems of units at compile-time as part of the static type checking phase without requiring modifications to the compiler or an additional tool. This is possible due to C++’s flexible type system offering templates and the on it building template metaprogramming technique [1], which allows to algorithmically transform and derive new types during compilation.

The library represents an arbitrary *composite unit* with the help of the template class `unit<Dim, System>`¹. Its two template arguments encode in static type lists the *dimension* as a reduced ordered set of *base dimensions* raised to a *rational power* and the associated *system of units* that defines the set of base dimensions and their respective *measures*. For example, the energy is represented through $[M]^1[L]^2[T]^{-2}$ using the base dimensions mass [M], length [L], and time [T], which is in SI units: $\text{kg m}^2 \text{s}^{-2} = \text{N m} = \text{J}$. The compile-time derivation of new units due to arithmetical operations with units, i.e., the dimensional analysis, is done through traits classes². The unit type `U` and value type `V` (by default `double`, but maybe, e.g., `std::complex<T>`) form a unique type `quantity<U, V>`, which overloads only the legal arithmetic and assignment operators. Thus, the compiler issues a “missing overload” error for illegal operations, e.g., the sum of two quantities with different units or the assignment between incompatible quantities. Numerical constants are annotated in the source code by multiplying them with their measurement unit, e.g.:

```
1 quantity<si::energy> E = 1.5 * si::newton * si::meter;
```

where `si::newton` is a static constant of type `unit<force_dimension, si::system>` and `si::meter` is a static constant of type `unit<length_dimension, si::system>`. Their multiplication with a `double` value yields the correct type `quantity<si::energy, double>` for the assignment to the variable `E`. The type `si::energy` is itself a convenience `typedef` for `unit<energy_dimension, si::system>`, which is proposed by the Boost.Units library as part of the definition for the SI system of units. Let us now look on a more complex example involving three quantities of different dimension and different value type:

```
1 // Current quantity with double value.
2 quantity<si::current> i = 0.1 * si::ampere;
3 // Complex impedance quantity.
4 quantity<si::resistance, complex<double> >
5   Z = complex<double>(4.0, 3.0) * si::ohm;
6 // The product yields a complex voltage quantity:
7 quantity<si::electric_potential, complex<double> >
8   v = Z * i;
```

The multiplication of the scalar current quantity variable `i` with the complex impedance quantity variable `Z` yields a complex voltage quantity, which is assigned to the variable `v` of the correct type.

¹All types and functions of Boost.Units are defined in the namespace `boost::units`. Each system of units is implemented in an own namespace, e.g., `boost::units::si` for the SI system of units.

²Traits are a generic programming technique to associate additional information to a type by creating a template class, which is specialised for each type it provides information about. This is done in form of typedefs, static constants, and static member functions, which have in each specialisation of the class the same name. A generic algorithm can thus deduce any information it needs about a type for its operation from the traits class. The C++ standard provides, e.g., the `std::numeric_limits<T>` and `std::character_traits<T>` traits [163]. The Boost.TypeTraits library is another example [6]. More information about traits can be, e.g., found in Adobe Systems Inc. et al. [6], Maddock and Cleary [103], Myers [125], Veldhuizen [175], and Frogley [56].

Boost.Units also overloads all standard mathematical functions defined in `<cmath>` for the new quantity type taking into account any unit transformations by the functions. Thus it is possible to write, e.g.:

```
1 quantity<si::dimensionless>
2   sin_half_pi_rad = sin(0.5 * M_PI * si::radian); // == 1.0
3 double sin_90_deg = sin(90.0 * degree::degree);   // == 1.0
```

The result for both assignments will be $\sin(\frac{\pi}{2} \cdot \text{rad}) = \sin(90^\circ) = 1$. This shows that the trigonometric functions are able to distinguish the different units of the quantity constants passed as function arguments. Both quantity constants have the dimension of a plane angle, which is the dimension the sine function requires as argument. The sine function returns a dimensionless quantity, which is the only quantity type that can be implicitly converted to its value type (usually **double**). All three examples will compile without warning/error and give the expected results.

As the unit is encoded into the quantity only as part of its type and not as a member variable, modern C++ compiler can optimise this information away after the type checking phase. This optimisation leaves behind an object with the same memory layout as the value type. Thus, no runtime penalty is caused by doing arithmetics with quantities. Only the compile time is increased.

Now, let us have a look on the compiler errors resulting from illegal mathematical operations involving quantity types. In a first example, two variables with similar names, but completely different meaning, are summed:

```
1 quantity<si::electric_potential> v_1 = 5.0 * si::volt;
2 quantity<si::velocity>           v_2 = 3.0 * si::meter / si::second;
3 // This won't compile:
4 quantity<si::electric_potential> v_tot = v_1 + v_2;
```

The compiler will refuse to compile the last statement and give an error³ like this:

```
1 fail_units_examples.cpp:16: error: no match for 'operator+' in 'v_1 + v_2'
```

This clearly points us to the error, as it is not allowed to sum a voltage quantity **v_1** and a speed quantity **v_2**. Especially in multiphysical models, it is common to meet such kind of errors, as different engineering domains have partially overlapping symbol standards, like in our case for voltage and velocity. Unfortunately, the generated compiler errors involving the Boost.Units quantity are not always so clear and compact. To give an example, let us implement an incorrect Ohm's law:

```
1 quantity<si::resistance>      R = 5.0e3 * si::ohm;
2 quantity<si::electric_potential> v = 10.0 * si::volt;
3 // This won't compile:
4 quantity<si::current>        i = R * v;
```

This simple mistake gives the incomprehensible compiler error message shown in Listing 5.8. The message can be made more comprehensible by removing the `boost::units::` namespace qualifiers and correctly indenting the mentioned template types, as shown in Listing 5.9. In it, the compiler states that it cannot convert from $[\Omega V] = [\text{m}^4 \text{kg}^2 \text{s}^{-6} \text{A}^{-3}]$ to $[\text{A}]$. Passing a value of the wrong quantity type as an argument to a function will cause similar type conversion error messages or no matching function call error messages.

The given code examples show that the complex infrastructure of types encoding all properties of a quantity (dimension, system of units, value type) is mostly hidden from the user while he is writing his code. The syntax for writing equations using quantity variables and quantity constants is equivalent to the mathematical one. Only the identifiers for the measurement units are longer. The typing overhead compared to normal C++ code using only **double** as type for the variables is negligible, as comments

³The given examples of compiler messages were generated by GNU g++ 4.2.1.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Listing 5.8: Unformatted compiler error message for “quantity<si::current> i = R * v;”.

```

1 fail_units_examples.cpp:10: error: conversion from ‘boost::units::quantity< boost::
  ::units::unit<boost::units::list<boost::units::dim< boost::units::
  length_base_dimension, boost::units::static_rational<4l, 1l> >, boost::units::
  list<boost::units::dim<boost::units::mass_base_dimension, boost::units::
  static_rational<2l, 1l> >, boost::units::list<boost::units::dim<boost::units::
  time_base_dimension, boost::units::static_rational<-0x00000000000000006l, 1l>
  >, boost::units::list<boost::units::dim<boost::units::current_base_dimension,
  boost::units::static_rational<-0x00000000000000003l, 1l> >, boost::units::
  dimensionless_type> > > >, boost::units::homogeneous_system<boost::units::list<
  <boost::units::si::meter_base_unit, boost::units::list<boost::units::
  scaled_base_unit<boost::units::cgs::gram_base_unit, boost::units::scale<10l,
  boost::units::static_rational<3l, 1l> > >, boost::units::list<boost::units::si::
  second_base_unit, boost::units::list<boost::units::si::ampere_base_unit,
  boost::units::list<boost::units::si::kelvin_base_unit, boost::units::list<
  boost::units::si::mole_base_unit, boost::units::list<boost::units::si::
  candela_base_unit, boost::units::list<boost::units::angle::radian_base_unit,
  boost::units::list<boost::units::angle::steradian_base_unit, boost::units::
  dimensionless_type> > > > > > > >, void>, double>’ to non-scalar type ‘
  boost::units::quantity<boost::units::unit<boost::units::list<boost::units::dim<
  <boost::units::current_base_dimension, boost::units::static_rational<1l, 1l>
  >, boost::units::dimensionless_type>, boost::units::homogeneous_system<boost::
  units::list<boost::units::si::meter_base_unit, boost::units::list<boost::units::
  scaled_base_unit<boost::units::cgs::gram_base_unit, boost::units::scale<10l,
  boost::units::static_rational<3l, 1l> > >, boost::units::list<boost::units::
  si::second_base_unit, boost::units::list<boost::units::si::ampere_base_unit,
  boost::units::list<boost::units::si::kelvin_base_unit, boost::units::list<
  boost::units::si::mole_base_unit, boost::units::list<boost::units::si::
  candela_base_unit, boost::units::list<boost::units::angle::radian_base_unit,
  boost::units::list<boost::units::angle::steradian_base_unit, boost::units::
  dimensionless_type> > > > > > > >, void>, double>’ requested

```

Listing 5.9: Simplified compiler error message for “quantity<si::current> i = R * v;”.

```

1 fail_units_examples.cpp:10: error: conversion from
2 ‘quantity<unit<list<dim<length_base_dimension, static_rational<4l, 1l> >,
3     list<dim<mass_base_dimension, static_rational<2l, 1l> >,
4     list<dim<time_base_dimension,
5         static_rational<-6l, 1l> >,
6     list<dim<current_base_dimension,
7         static_rational<-3l, 1l> >,
8         dimensionless_type> > > >,
9     si::system, void>,
10     double>’
11 to non-scalar type ‘quantity<si::current, double>’ requested

```

clarifying the involved units can be avoided when using the new quantity type. However, compiler errors can get very cryptic, because they usually state the fully expanded type names. This is a grave usability problem, which will hinder the acceptance of the proposed quantity types in the AMS design community. Therefore, it needs to be addressed.

5.4.2. Facilitating the Debugging of Errors Related to Quantity Types

The problem of incomprehensible compiler messages due to long template type names is known in the C++ community since the advent of generic libraries (e.g., the Standard Template Library (STL) [163]). To alleviate the problem, filters have been developed, which parse and simplify the error messages output by the C++ compiler. One example is STLfilt [183, 184]. It performs basic substitutions (using regular expressions) for all the standard STL components. It has been adapted to different compilers so that certain versions of it go further with respect to message ordering, line wrapping, library header error treatment, etc. Another example is TextFilt [55], which also applies regular expressions to the read in compiler output to do substitutions. It is a bit more generic than STLfilt, as the substitution process is controlled by a set of user-defined rules. However, both tools will yield in the best case a result close to the hand-formatted error message shown in Listing 5.9, as they lack an understanding of the information encoded into the quantity type.

As can be seen in the example error message, it is the `boost::units::unit<Dim, System>` type, which contributes to most of the “noise” in the message, but also contains the essential information to resolve the error in the user’s code. Therefore, to achieve even better results than the abovementioned generic tools, the `boost::units::unit<Dim, System>` types appearing in the compiler error messages need to be properly parsed to transform the contained information into a human-readable format. This has been realised in the `bufilt` (Boost.Units filter) utility implemented by the author of this Ph.D. thesis. Figure 5.9 depicts its architecture. The main component controlling the process is the *C++ template type filter*, which task it is to read the compiler log from the standard input and localise in it all occurrences of `boost::units::unit<...>`, to reformat them before they are written to the standard output. All text not belonging to the searched type is directly forwarded to the standard output. Thus, the parser of the reformatter is not disturbed by unexpected input. To this end, the filter successively reads, controlled by a Finite State Machine (FSM), characters from the standard input and stores them in a FIFO character buffer. For each read character *c*, the FSM decides if it is part of the searched template type `boost::units::unit<...>` or not. In the latter case, it flushes the buffer directly to the standard output. Once the whole searched type with all its template argument has been read into the character buffer, the filter applies the *reformat unit action* to it, which can be configured by the user through a command line argument that specifies the format for the output unit. Afterwards, the transformed type string is flushed to the standard output.

The reformatting of the unit type requires two steps. First, the *unit type parser* analyses, with the help of a grammar for the unit template type, the string in the character buffer to decompose it into an *unit object*. This object hierarchically contains all the information that have been encoded into the `boost::units::unit<...>` template type as a one-to-one mapping. Figure 5.10 shows the data model of this unit object. In it, the measurement unit is represented as a *dimension* and a *system of units*. The *dimension* is represented as a list of *base dimensions* raised to a *rational power*. The system of units can be either homogeneous or heterogeneous. In a *homogeneous system of units*, there is a one to one mapping between the *base dimensions* and the *scaled base units*. In a *heterogeneous system of units*, at least for one base dimension two or more different scaled base units have been specified. Therefore, the heterogeneous system has to directly encode the measurement unit representing the dimension of

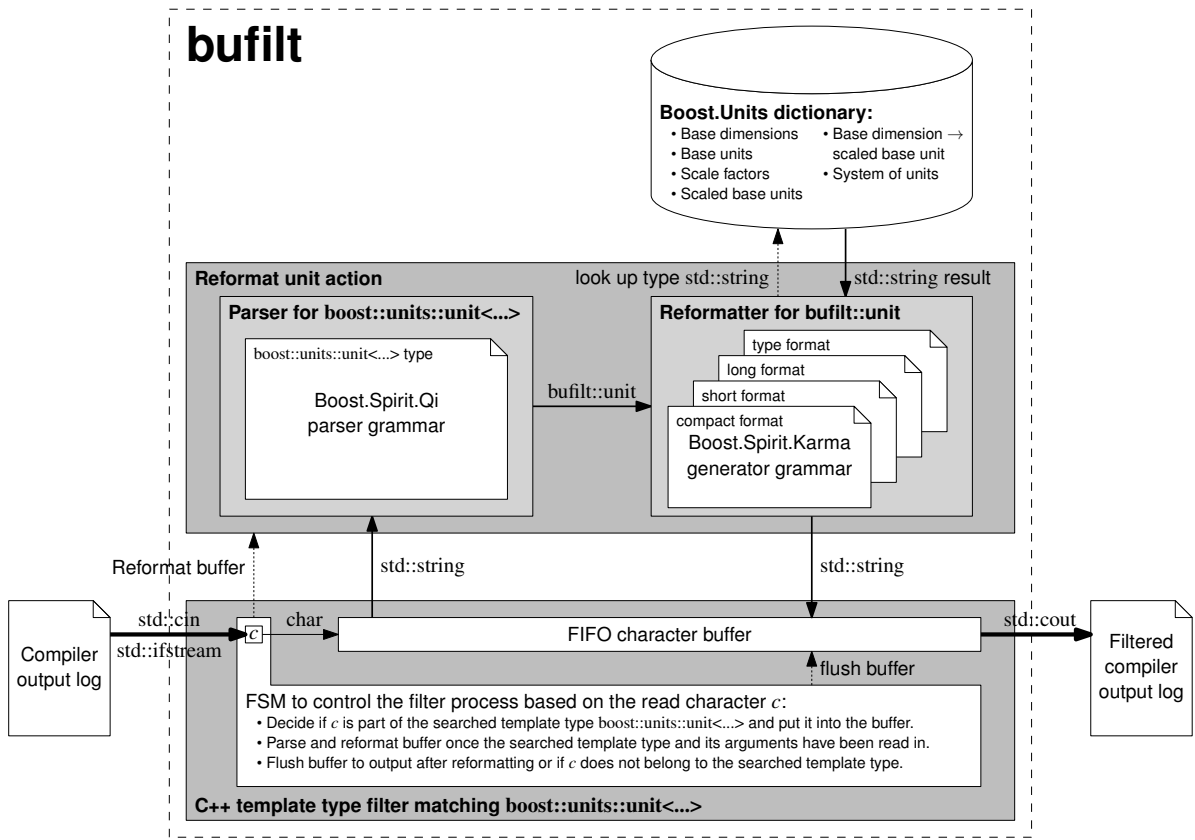


Figure 5.9.: Architecture of the bufilt utility.

the unit. This is done as a list of *scaled base units* raised to a *rational power* via the intermediate type *heterogeneous_system_dim*. Scaled base units and heterogeneous systems can have an associated *scale factor* of an *integer base* raised to a *rational power*. This complex data model is necessary to represent arbitrary measurement units in arbitrary systems of units. It explains the complexity of the unit template types, which is further complicated by the fact that the various lists need to be represented as recursive template type instances.

In the second step, the unit object returned by the parser is transformed back into a string by the *reformatter*. It uses a *generator grammar* with attached actions for the configured output format to transform the individual attributes of the unit object into strings and arranging them to the reformatted unit type. Currently, four output formats (“compact”, “short”, “long”, and “type”) for the unit are provided and will be explained below together with the usage of the tool. This reformatting process is supported by the *Boost.Units dictionary*. In it, the reformatter can look up the names and symbols of base dimensions, base units, scale factors, and scaled base units based on the passed type name. Additionally, this dictionary contains the mapping of base dimensions to scaled base units and allows to identify common system of units (e.g., SI) based on the supplied list of scaled base units defining it. Currently, the dictionary contains all the information about the base dimensions, base units, systems of units, etc., which are predefined in the Boost.Units 1.1.0 library [158]. This covers the SI and Centimetre Gram Second (CGS) systems of units as well as many common astronomical, imperial, temperature, and angle units. In the relatively infrequent case that a user defines his own base dimensions, base units, system of

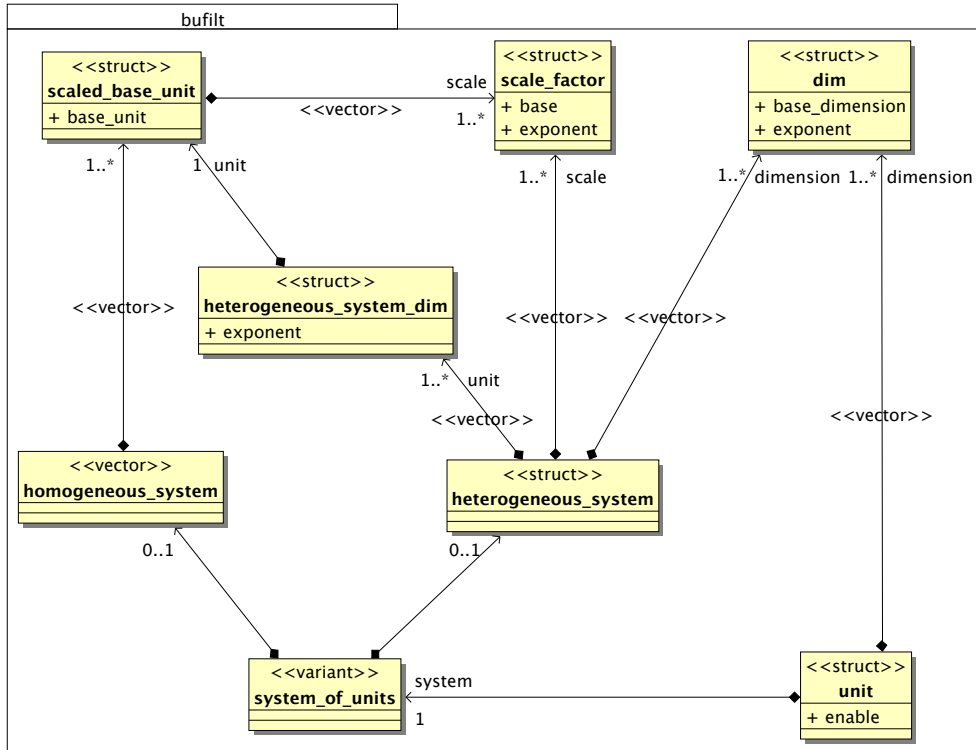


Figure 5.10.: Class diagram of bufilt's data model to represent a unit.

units, etc., he will have to add the information about them to the source file of the Boost.Units dictionary and recompile bufilt. This decision has been deliberately taken to provide a simple tool in form of a stand-alone executable that does not depend on any external files to function.

The implementation of bufilt in C++ makes only use of the C++ standard library and Boost libraries [41]. Therefore, it does not impose additional dependencies than already set by the usage of the Boost.Units library for dimensional analysis. The C++ template type filter is implemented as a filtering input/output stream using Boost.Iostreams [167]. The parser and generator of the reformat unit action have been implemented using Boost.Spirit [43], which is an object-oriented, recursive-descent parser and output generation library for C++. It allows to write grammars and format descriptions using a syntax similar to the Extended Backus-Naur Form (EBNF) [88] directly in C++. These inline grammar specifications can directly include C++ code to specify actions. Thus, it is relatively easy to build and handle the hierarchical data structure of the unit object discussed above. The Boost.Program_options library [149] has been used to implement the handling of the bufilt command line options.

The bufilt utility can either read log files generated by the C++ compiler or become part of a command line pipe to directly process the compiler output. In the latter case, it is important to redirect first stderr to stdout using `2>&1` (bash syntax), before piping it into the bufilt utility. The bufilt utility supports the following command line options:

```

-f [ --format ] arg (=compact) : Select the unit output format ("compact", "long", "short",
                                "type").
-h [ --help ]                  : Show the help message.
-v [ --version ]               : Show the program version.
  
```

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

On the command line, it can be executed in the following ways:

```
$ bufilt [-h]
$ bufilt [-v]
$ bufilt [-f arg] file [file ...] [> output_file]
$ bufilt [-f arg] < input_file [> output_file]
$ c++ ... 2>&1 | bufilt [-f arg] [> output_file]
```

To show the effect of the four different unit output formats, the compiler error message caused by the wrong implementation of Ohm's law will be used again (Listing 5.8). The **type** format is the closest to the original output. All type names encoding base dimensions or base units are conserved. Only the types encoding the type lists, scale factors, and powers are replaced by a more readable notation:

```
1 fail_units_examples.cpp:10: error: conversion from 'boost::units::quantity<boost::units::unit<{boost::units::length_base_dimension^4, boost::units::mass_base_dimension^2, boost::units::time_base_dimension^-6, boost::units::current_base_dimension^-3, boost::units::dimensionless_type}, boost::units::si::system>, double>' to non-scalar type 'boost::units::quantity<boost::units::unit<{boost::units::current_base_dimension, boost::units::dimensionless_type}, boost::units::si::system>, double>' requested
```

Additionally, **bufilt** tries to identify the system of units based on the scaled base units list of homogeneous systems and to replace the lengthy list with the corresponding **typedef** provided by the Boost.Units library. As this format does not involve any look-up in the Boost.Units dictionary, it will always work even in the presence of user-defined base units and base dimensions.

The long format additionally replaces the type names representing base dimensions, scaled base units, and system of units with their regular name:

```
1 fail_units_examples.cpp:10: error: conversion from 'boost::units::quantity<boost::units::unit<{length^4, mass^2, time^-6, (electric current)^-3, dimensionless}, SI>, double>' to non-scalar type 'boost::units::quantity<boost::units::unit<{electric current, dimensionless}, SI>, double>' requested
```

The **short** format is very similar. It replaces the type names by their corresponding symbols:

```
1 fail_units_examples.cpp:10: error: conversion from 'boost::units::quantity<boost::units::unit<{[L]^4, [M]^2, [T]^-6, [I]^-3, [1]}, SI>, double>' to non-scalar type 'boost::units::quantity<boost::units::unit<{[I], [1]}, SI>, double>' requested
```

This format is already quite readable and still conserves all the information encoded into the unit type.

Finally, the (default) **compact** format goes even one step further. It maps the base dimensions on their corresponding base units and formats the measurement unit and system of units as compact human-readable strings:

```
1 fail_units_examples.cpp:10: error: conversion from 'boost::units::quantity<boost::units::unit<"m^4 kg^2 s^-6 A^-3", "SI">, double>' to non-scalar type 'boost::units::quantity<boost::units::unit<"A", "SI">, double>' requested
```

This format makes it most easy to the user to understand the compiler error messages and, therefore, is most suitable for day-to-day use.

The **bufilt** utility implements a full parser for the `boost::units::unit<Dim, System>` type, which accepts spaces and line breaks at the same places as they are permitted in C++. However, it has been only tested with error messages generated by GNU's C++ compiler `g++` in version 4.2.1. Therefore, it might fail to reformat messages generated by other compilers—especially, if these compilers substitute on their side already parts of the `boost::units::unit<...>` type instantiations.

With the presented `bufilt` utility at hand, using the Boost.Units library in real-world projects becomes more practical. Its usage will have a positive impact on the code quality, as the designer/programmer can specify much more precisely the interfaces and computations in his model/programme, which involve physical quantities. The compiler can check the coherency of the connected interfaces and implemented equations. Thus, many problems can be localised and fixed before the first execution of the model/programme. This gives the designer/programmer more time to test the important behavioural aspects of his model/programme.

5.4.3. Using Quantity Types in SystemC Models

The basic integration of the Boost.Units library with SystemC and its AMS extensions is relatively easy. The Boost.Units quantity types can be directly used for computations inside any SystemC module. To use them in the modules' port interfaces and as a data type for the signals bound to the ports, the requirements to the data type for usage in signals set forth by the SystemC LRM [79, clause 6.4.3] and the AMS extension LRM [131, clause 4.1.1.3.3] have to be satisfied.

The Boost.Units quantity type directly satisfies the essential requirements: it implements the equality **operator**==, the output stream **operator**<<, the assignment **operator**=, and the default constructor. Therefore, it can be directly used as data type for DE signals (e.g., `sc_core::sc_signal<T>`) and DE ports (e.g., `sc_core::sc_in<T>` or `sc_core::sc_out<T>`) as well as for TDF signals (`sca_tdf::sca_signal<T>`) and TDF ports (e.g., `sca_tdf::sca_in<T>`, `sca_tdf::sca_out<T>`).

Only the tracing mechanisms of SystemC and its AMS extensions need to be customised to enable the tracing of quantity ports and signals. For SystemC, this is possible in a straightforward way by providing the necessary overload of the `sc_core::sc_trace()` function for a constant reference to a quantity value. Please note that only the value part of the quantity should be written to the trace file and not the measurement unit. To this end, the quantity type offers the member function `boost::units::quantity<U, T>::value()`, which returns the required reference. For the AMS extensions, there is a slight complication: they impose the usage of the output stream operator for the writing of trace values in time-domain simulations [131, clause 4.1.1.3.3]. However, this also causes the measurement units to appear in the trace file *for each written value* instead of only in the header of the VCD and of the tabular trace files. This breaks the defined file format so that the files cannot be opened anymore by the waveform viewers. It is not sensible to locally patch the implementation of the Boost.Units library for this special purpose. Unfortunately, the System AMS extensions 1.0 LRM does not define a type-specific customisation mechanism allowing to reformat the trace values. Therefore, a vendor-specific solution for the used Fraunhofer SystemC-AMS library had to be found. Fortunately, this did not require patching the SystemC-AMS library itself. It sufficed to partially specialise two internal template classes for the `boost::units::quantity<U, T>` type, which are defined in the `sca_util::sca_implementation` namespace: `sca_trace_value_handler<T>` and `sca_type_explorer<T>`. This can be done in an external header that is included additionally to the "systemc-ams" header file in the programme. Unfortunately, this does not present a good solution for standardisation: too many implementation details of SystemC-AMS had to be copied into this new header file, which are unrelated to the reformatting of the quantity type (e.g., in the case of `sca_trace_value_handler<T>`, ca. 120 lines of internal code had to be copied to modify one single line). This solution is still better than the only standard-conforming solution of implementing a wrapper class for the `boost::units::quantity<U, T>` type for the sole purpose of redefining its output stream **operator**<<. The latter would have been heavier without any added benefit for the user. He usually wants to have the quantity value and measurement unit output

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

together when printing a *single* quantity value.

With this implementation-specific customisation in place, there are no more noticeable differences between `boost::units::quantity<U,T>` and standard C++ or SystemC data types like **double** or `sc_dt::sc_int<N>` when writing SystemC (AMS) models.

This tracing customisation problem does not only concern the quantity data type, but also other user-defined data types, which require different output formats for single values and tabulated values. The SystemC solution of providing an overload to `sc_core::sc_trace()` is in the opinion of the author not generic enough: It only allows the decomposition of **structs** of standard C++ or SystemC data types into individual traces. It does not allow the “on-the-fly” transformation of a user-defined data type into a data type supported by SystemC or the AMS extensions for tracing. This limitation requires the user to add artificial data members or ports to his modules, which he needs to keep in sync with the actual data for the sole purpose of tracing. One solution could be to encapsulate all information (e.g., data type size, equivalent data type in the trace file, measurement unit) and operations (e.g., to write a correctly formatted value into the trace file) needed by the tracing mechanisms about/on a data type into a traits class⁴, which could be specialised for each data type to be traced. The `std::char_traits<T>` class of C++ is an example for such a mechanism, which encapsulates all character-specific information and operations required by the `std::basic_string<T>` for the implementation of strings of different character bit width [163]. It would be nice if the tracing customisation problem were addressed in one of the next standard releases of the AMS extensions.

5.5. SystemC AMS extensions eXperiments Library

The described integration of the Boost.Units library with the SystemC AMS extensions (Section 5.2) has been the starting point for the development of the SystemC AMS extensions eXperiments (SCAX) library. Its mission, within the SystemC AMS simulation framework, is to facilitate the modelling of heterogeneous multiphysical systems in a formal and consistent way on high levels of abstraction. To this end, it supports the *bond graph formalism* (Section 5.3.2) for the description of conservative (energy conserving) behaviour and the *block diagram formalism* (Section 5.3.3) for the description of non-conservative (analogue signal processing) behaviour. Both formalisms can be mixed using the new Bond Graph (BG) MoC, which can directly interact with the DE and TDF MoCs of the SystemC AMS extensions. It has been shown in Sections 5.3.2 and 5.3.3 that the expressiveness of these two formalisms stems from their generic modelling primitives, which behaviour can be highly parameterised. Both sections also showed how important it is to be able to precisely specify the physical domain or quantities at the primitives’ ports to ensure the correct assembly of the models for the targeted multiphysical system. In order to not lose these advantages, the SCAX library provides for each primitive a generic module, which port interface and behaviour can be parameterised in a well-defined way. To achieve this flexibility, the library design has been inspired by the generic and functional programming techniques used in the STL [89, 163] and Boost libraries [42].

The library offers its functionality in three thematic headers:

scax_utility: Utilities for the SystemC AMS extensions, which are not linked to a specific MoC but facilitate the development of generic modules (cf. to Section 5.6), which can seamlessly use the C++ standard data types, SystemC data types, and Boost.Units quantity data types. This includes:

⁴See footnote 2 on page 92 for an explanation of the traits technique.

- Cast functions to uniformly convert (back and forth) between the SystemC AMS extensions' time types (`sc_core::sc_time` and `sca_core::sca_time`) and a user-specified data type `T` (e.g., `double` or `quantity<si::time>`).
- Data type traits to deduce from a (quantity) type, the type of its value, unit, and system of units as well as a compatible time type.
- Waveform functors⁵ (cf. to Table A.2 on page 152) to generate standard continuous-time stimuli like constant, exponential, pulse, sinusoidal signals.
- Functors for testing data sequences against a condition (e.g., threshold crossing) and reacting on a change of the condition result.
- Lazily evaluated function wrappers for the mathematical functions defined in `<cmath>` and `<boost/units/cmath.hpp>` to facilitate their usage in functors created with the Boost.Lambda library [84].

All utilities are implemented in the namespace `scax_util`.

scax_tdf: A library of generic block diagram modules (cf. to Section 5.6 and Table A.4 on page 154) for the TDF MoC, which port data types and behaviour can be parameterised. All modules are implemented in the namespace `scax_tdf`.

scax_bond_graph: A new Bond Graph (BG) MoC (cf. to Section 5.7), which supports mixed bond graph (conservative) / block diagram (non-conservative) models of heterogeneous multiphysical systems. It provides a library of predefined generic bond graph and block diagram primitive modules (cf. to Tables A.5 and A.6 on pages 156 and 157, respectively). The user can implement his own bond graph and block diagram modules. The BG MoC implements synchronisation with the TDF and DE MoCs. It is implemented in the namespace `scax_bg`.

More details on selected aspects of the library will be given in the following sections. Appendix A provides the reader with a short reference of the SCAX library. The SCAX library itself is fully documented using Doxygen [173].

To be portable, its implementation tries to rely as much as possible on the functionality provided by the C++, SystemC, and AMS extensions standards [79, 89, 131]. However, the already described integration with the Boost.Units library (Section 5.4.3) and the integration of the BG MoC via the synchronisation layer into the SystemC AMS extensions (Figure 5.1 on page 74), required the use of several implementation-specific APIs due to a lack of standardisation of these aspects. This currently links the SCAX library to the OSCI SystemC 2.2 reference implementation [82] and the Fraunhofer SystemC-AMS PoC implementation [53] of the abovementioned IEEE/OSCI standards. Besides these two libraries, the SCAX library implementation makes use of several Boost libraries [42], which does not affect the portability, as they purely use features of the ISO/IEC C++ standard [89]. All development and testing has been carried out on Linux (Debian Lenny, Linux kernel 2.6.32, GNU C++ compiler g++ 4.4.5) and Mac OS X (versions 10.5 and 10.6, GNU C++ compiler g++ 4.2.1) on the i386 and x86_64 processor platforms.

⁵Functor is the common term for a function object, i.e., an object of a class that overloads the function call `operator()`. The advantage of functors over ordinary functions is that they can carry over state from one function call to another.

5.6. Generic TDF Modules for Common Block Diagram Primitives

In a first step towards the modelling of heterogeneous multiphysical systems (Section 5.3) with the SystemC AMS extensions (Section 5.2), the block diagram formalism has been used with the TDF MoC (Section 5.2.1). At a first glance, the ELN and LSF MoCs (Section 5.2.2) might seem to be more appropriate choices. However, they are limited to the description of purely linear conservative and non-conservative behaviour and therefore not well suited for the description of the nonlinear behaviour of multiphysical systems such as the electromechanical transducer example discussed in Section 5.3. Additionally, only the DE and TDF MoCs allow us to parameterise their ports and signals to use the Boost.Units quantity type presented in Section 5.4. This is important for a precise specification of the module interfaces that keeps the link with the physical domain despite the high level of abstraction of the block diagram formalism (Section 5.3.3). The usage of the quantity types enables the compiler to check the correct assembly of the model and the coherency of the implemented computations.

However, there is currently no standard library of TDF modules for the block diagram primitives available, which fulfils the needs of parameterisable ports and behaviour described in the previous section. For this reason models with quite similar generic behaviour (e.g., sources, amplifiers, functional blocks) are often reimplemented to cope with small differences in the involved data types or behaviour, which is against the model reuse idea. This situation is aggravated by the introduction of quantity types, which purposely tie a model even further to a specific application. This motivated the development of the `scax_tdf` part of the SCAX library, which offers a set of generic TDF modules for common block diagram primitives like source, sink, summer, multiplier, function, integrator, and differentiator blocks.

5.6.1. Implementation

Table A.4 in Appendix A summarises all available modules. As the function carried out by each block diagram primitive is orthogonal to the actual type of the inputs and outputs, each TDF module is implemented as a template class, which allows to parameterise its port and parameter types upon instantiation. Dependencies between these types, due to the computations implemented by the module, are either automatically satisfied or at least checked for consistency using traits classes or static assertions, respectively. These new TDF modules are not constrained to work exclusively with `boost::units::quantity<U, V>` types, but can also be parameterised for other types such as **double**. This ensures interoperability with other user-created TDF modules. To render the library even more flexible, some modules, such as the source module and the various function modules, take a function as argument upon instantiation. Thus the user can specify, e.g., the waveform to be generated by the source instance or the function to be applied to the read input samples to transform them to an output sample by the function block instance.

The time-dependent function module with two inputs (`scax_func2t<T1, T2, T3, TimeType>`) serves as a good example to show how such generic TDF modules are implemented. Its implementation is given in Listing 5.10. The user can specify, via the template parameters, the data types of the input (T1, T2) and output (T3) ports as well as the data type to represent the time (TimeType). Usually, the TimeType can be derived from the port data types (in our case T1). This is done via its default value, which is the traits class `scax_util::scax_data_type_traits<T1>` that implements the transformation of T1 to a compatible time data type. For example, if `T1 = double`, then the traits will return also **double** as the time type. If T1 is a quantity using the SI system of units (e.g., `T1 = quantity<si::current>`), then the traits will return the matching quantity time type `quantity<si::time>`. Part of the public interface of each implemented TDF module are **typedefs** for the data types of each port, parameter, and

function wrapper, which are subject to change due to the template parameterisation of the module. This allows to reference these derived data types inside and outside the module via a unique identifier, e.g., to declare the interfacing signals in the code instantiating the module or to calculate the parameters for the module instance, even if their exact types are yet to be fixed by the compiler based on the template parameters. The class constructor `scax_func2t()` initialises, upon module instantiation, the ports with their instance names and the function wrapper `f_` with the user-supplied sample transformation function `f`. The function wrapper of type `std::tr1::function<...>` can accept any free function or functor matching the specified interface of argument and return types and is thus more flexible than classical function pointers [90]. The `set_attributes()` member function, as defined in the standardised SystemC AMS extensions, specifies at elaboration that at each simulated time step one sample is read from each input port and one sample is produced at the output port. Finally, the `processing()` member function implements the module's behaviour by reading samples from the input ports, applying the transformation function, and writing the returned result to the output port each time it is called by the SystemC AMS extensions for a new time step.

This example showed all principle elements of a generic TDF module, which ports and behaviour can be parameterised. The **typedefs** at the beginning of the module implementation are the central place to construct new types based on the template arguments by passing them as arguments to other template classes or traits. Thus, all dependencies between the types, due to the computations implemented in the model, can be expressed. The rest of the module implementation is very similar to the TDF modules already presented in Section 5.2.1. All other generic TDF modules from the `scax_tdf` library follow the same structure.

5.6.2. Application Example

As an application example, the block diagram model of the electromechanical transducer linked to a micromechanical resonator (Section 5.3.3, Figure 5.5e) has been implemented using the developed `scax_tdf` library. Figure 5.11a shows the schematic of the equivalent TDF model. The topological loops in the block diagram need to be broken for the TDF model by assigning a one step delay to the integrator outputs.

The transducer is encapsulated in the SystemC module `elmech_transducer`, which is derived from class `sc_core::sc_module` (Listing 5.11). The transducer module can be parameterised upon instantiation with the voltage and force transfer functions (`v_func` corresponding to $\frac{q}{C(x)}$ and `F_func` corresponding to $F(q, x)$ in Figure 5.11a, respectively) by passing them along with the initial conditions as constructor arguments. Compared to the purely computational data type **double**, the usage of quantity types strengthens the signature of the argument list. In this way, the compiler can detect if parameters are accidentally passed in the wrong order, e.g., for the initial charge `q_0` and displacement `x_0` of the transducer. As the transducer model is entirely structural, it contains only block diagram component instances and specifies their interconnection by binding their ports to signals in the constructor. **typedefs** are used to facilitate the usage of the different `quantity<U, V>` types (in our case, e.g., `voltage_type`, `force_type`). The specification of the quantity type for the ports `v_out` and `v_in` make them recognisable as a voltage output and speed input, respectively, despite the overlapping symbols and without the need of comments. The compiler detects any binding of incompatible quantity signals to ports. This would not be the case if only the computational data type **double** would have been used for the signals and for the ports.

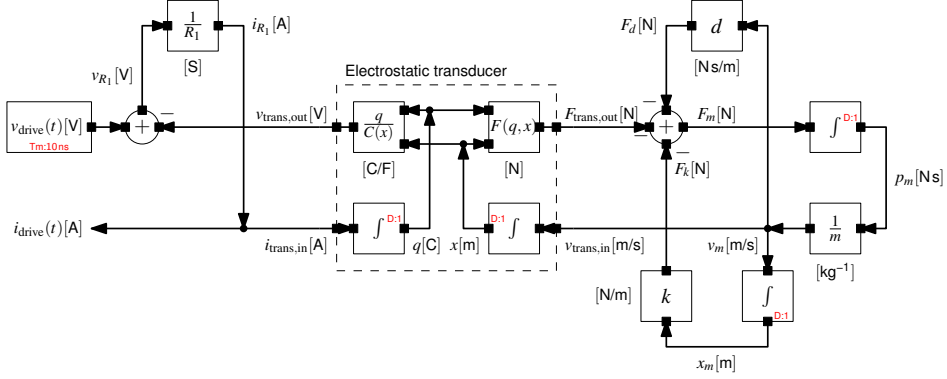
Listing 5.12 shows code extracts from the test bench for the transducer and resonator. Being also a structural model, it follows the same approach as the `elmech_transducer` model. What is new is

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

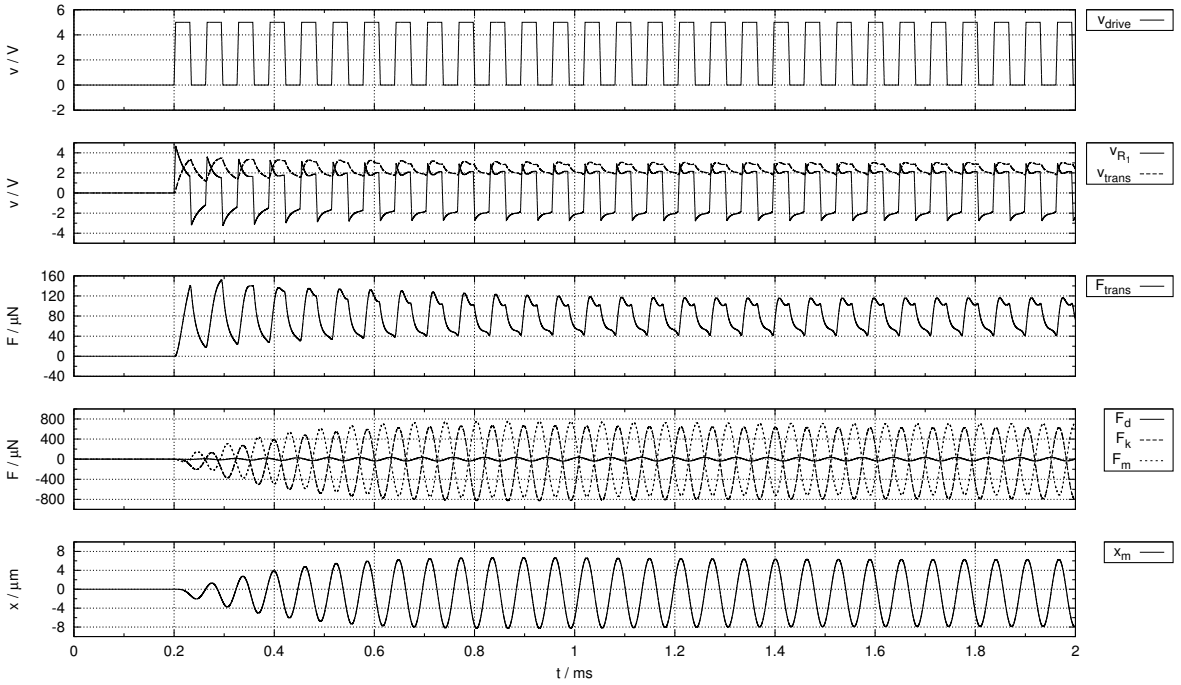
Listing 5.10: Time-dependent function module with two inputs from the scax_tdf library.

```
1  #ifndef SCAX_TDF_SCAX_FUNC2T_H
2  #define SCAX_TDF_SCAX_FUNC2T_H
3  #include <functional>
4  #include "systemc-ams"
5  #include "scax_tdf_common.h"
6  namespace scax_tdf {
7
8  // Generic TDF function module with two input ports.
9  //
10 // The module processes the read input samples to the output samples using a
11 // user supplied time-dependent function T3 f(T1, T2, TimeType).
12 template<typename T1, typename T2, typename T3,
13         typename TimeType =
14         typename sca_util::sca_data_type_traits<T1>::time_type>
15 class scax_func2t : public sca_tdf::sca_module {
16 public:
17     // Typedefs for each template parameter.
18     typedef T1 in1_data_type; // First input port data type.
19     // ...
20     // Type of the function wrapper transforming the input to the output samples.
21     typedef std::tr1::function<T3 (const T1&, const T2&, const TimeType&)>
22         function_type;
23
24     sca_tdf::sca_in<in1_data_type> in1; // First input port.
25     sca_tdf::sca_in<in2_data_type> in2; // Second input port.
26     sca_tdf::sca_out<out_data_type> out; // Output port.
27
28     // Initialises the generic TDF function module with a module name and
29     // signal processing function.
30     explicit scax_func2t(const sc_core::sc_module_name& name,
31                         const function_type& f)
32     : in1("in1"), in2("in2"), out("out"), f_(f)
33     {}
34     // ...
35 protected:
36     // Sets the port rates to one.
37     virtual void set_attributes() {
38         in1.set_rate(1); in2.set_rate(1);
39         out.set_rate(1);
40     }
41
42     // Processes the input samples through the stored function and writes the
43     // result to the output port.
44     virtual void processing() {
45         out.write(f_(in1.read(), in2.read(),
46                     sca_util::sca_time_cast<TimeType>(in1.get_time())));
47     }
48 private:
49     // Function wrapper for the transformation of the input to the output samples.
50     function_type f_;
51 }; // class scax_func2t<T>
52
53 } // namespace scax_tdf
54 #endif // SCAX_TDF_SCAX_FUNC2T_H
```

5.6. Generic TDF Modules for Common Block Diagram Primitives



(a) Schematic of the TDF block diagram model.



(b) Simulation results.

Figure 5.11.: Schematic and simulation results of the electromechanical transducer block diagram model for the TDF MoC. The topological loops need to be broken with unit delays at the integrator outputs. To reduce the numeric error during simulation to an acceptable level, the module time step needs to be reduced to 10 ns for the chosen parameterisation. The simulation performance is indicated in Table 5.4 on page 134.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Listing 5.11: Electromechanical transducer module using the scax_tdf library.

```
1 using namespace std; using namespace std::tr1;
2 using namespace boost::units; namespace si = boost::units::si;
3 using namespace scax_tdf;
4
5 // Block diagram model of an eletromechanical transducer.
6 class elmech_transducer : public sc_core::sc_module {
7 public:
8     // Typedefs for electrical and mechanical quantity types.
9     typedef quantity<si::electric_potential> voltage_type;
10    typedef quantity<si::force> force_type;
11    // ...
12
13    // Electrical and mechanical ports.
14    sca_tdf::sca_in<current_type> i_in;
15    sca_tdf::sca_out<voltage_type> v_out;
16    sca_tdf::sca_in<velocity_type> v_in;
17    sca_tdf::sca_out<force_type> F_out;
18
19    // Construct structural model of the transducer and initialise the force
20    // and voltage functions as well as the set the initial conditions.
21    elmech_transducer(
22        const sc_core::sc_module_name& name,
23        function<voltage_type (charge_type, displacement_type)> v_func,
24        function<force_type (charge_type, displacement_type)> F_func,
25        charge_type q_0 = 0.0 * si::coulomb,
26        displacement_type x_0 = 0.0 * si::meter)
27    : i_in("i_in"), v_out("v_out"), v_in("v_in"), F_out("F_out"),
28      q_sig("q_sig"), x_sig("x_sig"),
29      i_integ(new scax_integ_trapez<current_type>("i_integ", q_0)),
30      v_integ(new scax_integ_trapez<velocity_type>("v_integ", x_0)),
31      v_func2(new scax_func2<charge_type, displacement_type,
32                voltage_type>("v_func2", v_func)),
33      F_func2(new scax_func2<charge_type, displacement_type,
34                    force_type>("F_func2", F_func))
35    {
36        // Connectivity.
37        i_integ->in(i_in); i_integ->out(q_sig);
38        v_integ->in(v_in); v_integ->out(x_sig);
39        v_func2->in1(q_sig); v_func2->in2(x_sig); v_func2->out(v_out);
40        F_func2->in1(q_sig); F_func2->in2(x_sig); F_func2->out(F_out);
41    }
42
43 private:
44     // Internal signals.
45     sca_tdf::sca_signal<charge_type> q_sig;
46     sca_tdf::sca_signal<displacement_type> x_sig;
47
48     // Internal TDF modules.
49     auto_ptr<scax_integ_trapez<current_type> > i_integ;
50     auto_ptr<scax_integ_trapez<velocity_type> > v_integ;
51     auto_ptr<scax_func2<charge_type, displacement_type, voltage_type> > v_func2;
52     auto_ptr<scax_func2<charge_type, displacement_type, force_type> > F_func2;
53 }; // class elmech_transducer
```

the definition of quantity constants, which are used as instance parameters (e.g., R_1 , k). The usage of quantity types prevents any accidental assignment of a quantity constant of the right dimension but wrong system of units without proper conversion, e.g., the usage of pound instead of kilogram for the mass m . The transfer functions of the transducer module instance are realised as functors, which interfaces (types of arguments and return value) are defined in the template arguments of the function wrappers (`v_trans_func`, `F_trans_func`) [90]. The functors are defined in-place with the placeholders `_1` and `_2` representing the two arguments. This compact notation is made possible by the *Boost.Lambda* library [84]. The interface code between it and *Boost.Units* [158] was developed by the author of this Ph.D. thesis as part of the SCAX project and has become in the meantime official part of the *Boost.Units* library. Due to the usage of quantity types in the function interface, a contract is formed, which fulfilment by the user-supplied implementation can be checked by the compiler. Mixing up the order of the quantity arguments or forgetting a term in the implemented functions is detected in most cases due to the dimensional analysis, as it will usually lead to a conflict either on the level of mathematical operations or on the result type due to incompatible units. The waveform of the driving voltage source `v_drive_src` is specified slightly differently by passing a functor instance of type `scax_util::scax_pulse<T>` parameterised with the return type `voltage_type` and the parameters of the pulse waveform. Calling the functor with a time value as argument will return the waveform value at that point in time. `scax_util::scax_pulse<T>` is one of the several waveform generators available in the `scax_utility` library (Table A.2 on page 152).

Figure 5.11b shows the simulation results of the developed SystemC AMS model. The pulsed input voltage v_{drive} excites the sinusoidal oscillation of the mechanical resonator. Frequencies other than its natural resonance frequency are filtered out due to its high quality factor. The common mode of the driving voltage shows up in the resulting electrostatic force and displacement of the resonator. A variant of the model using **double** instead of `quantity<U, V>` yields, as expected, numerically the same results, however its code required much more explanatory comments to compensate for the semantical loss of quantity types and units. Reference models for the example developed in VHDL-AMS, once using branch quantities to describe it in form of a generalised network and once using free quantities to describe it in form of a block diagram, showed the same dynamic behaviour as the SystemC AMS models. This demonstrates the successful simulation of all major effects of this small system. Table 5.4 on page 134 compares the compile and execution times for the `quantity<U, V>` and **double** TDF model variants with the VHDL-AMS reference models and with the equivalent models using the bond graph and block diagram primitives of the new BG MoC presented in the next section. It can be seen that the usage of *Boost.Units* significantly increases the compile time, but hardly affects the simulation time. This is the price to pay for the achieved stronger checking of the model interfaces and equations. For the same simulation time step size of 10 ns, the TDF MoC outperforms the VHDL-AMS reference models (and BG MoC models). However, this comparison is not really fair, as the other models are able to converge reliably already with a maximum simulation time step size of 2 μ s and still yield a very similar precision in the simulation results. At this time step size, the VHDL-AMS (and BG MoC models) clearly outperformed the TDF models. With the same time step size, the TDF model did not yield any reasonable results: the oscillation amplitude grew exponentially and did not saturate. This is not surprising, because the TDF MoC purposely omits any error control and convergence iterations in favour of a higher simulation performance. Only by limiting the time step size to 10 ns, the simulation results were accurate enough to conform with the reference models.

The presented application example demonstrated the flexibility of the developed block diagram primitives library for the TDF MoC introducing dimensional analysis. The systematic usage of quantity types instead of **double** in the model leads to a more precise description of the system adding important

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Listing 5.12: Test bench for the electromechanical transducer and the mechanical resonator.

```
1 #include "systemc-ams"
2 // ... Definition of the elmech_transducer module...
3
4 int sc_main(int argc, char* argv[]) {
5     // Using and namespace aliases declarations, convenience typedefs, etc.
6     // ...
7     // Derivation of new units.
8     typedef divide_typeof_helper<si::force, si::length>::type stiffness;
9     typedef divide_typeof_helper<si::force, si::velocity>::type viscous_damping;
10    // Electrical component constants.
11    const quantity<si::resistance> R_1 = 50.0e3 * si::ohm;
12    const quantity<si::capacitance> C_trans_0 = 500.0e-12 * si::farad;
13    const quantity<si::length> overlap = 20.0e-6 * si::meter;
14    // Mechanical component constants.
15    const quantity<si::mass> m = 10.0e-9 * si::kilogram;
16    const quantity<stiffness> k = 100.0 * si::newton/si::meter;
17    const quantity<viscous_damping> d = 50.0e-6 * si::newton*si::second/si::meter;
18
19    // Transducer capacitance formula v_trans(q, x).
20    function<voltage_type (charge_type, displacement_type)>
21        v_trans_func = _1 / (C_trans_0 * (1.0 - (_2 / overlap)));
22    // Transducer force formula F_trans(q, x).
23    function<force_type (charge_type, displacement_type)>
24        F_trans_func = (_1 * _1 * overlap)
25                        / (2.0 * C_trans_0 * (overlap - _2) * (overlap - _2));
26
27    // Initial conditions and stimuli.
28    // ...
29    // Signals.
30    sca_tdf::sca_signal<voltage_type> v_drive("v_drive"), v_trans("v_trans");
31    // ...
32    sca_tdf::sca_signal<momentum_type> p_m("p_m");
33    sca_tdf::sca_signal<displacement_type> x_m("x_m");
34    // Electrical driving circuit.
35    scax_source<voltage_type>
36        v_drive_src("v_drive_src",
37                    scax_pulse<voltage_type>(V_drive_0, V_drive_1, t_drive_delay,
38                                            t_drive_rise, t_drive_fall, t_drive_pulse, t_drive_period));
39    v_drive_src.out(v_drive);
40    // ...
41    // Transducer.
42    elmech_transducer transducer("transducer", v_trans_func, F_trans_func,
43                                q_trans_0, x_m_0);
44    transducer.i_in(i_R_1); transducer.v_out(v_trans);
45    transducer.v_in(v_m); transducer.F_out(F_trans);
46    // Mechanical resonator.
47    // ...
48    // Tracing and simulation.
49    // ...
50    v_drive_src.set_timestep(t_step);
51    try { sc_start(t_sim); } catch (const exception& e) { cerr<<e.what()<<endl; }
52    // ...
53    return sc_core::sc_report_handler::get_count(sc_core::SC_ERROR);
54 }
```

semantical information about the used measurement units in a compact notation that the compiler can use for consistency checks. The simulation performance of the example also showed that the C++ compiler is able to optimise the computations involving quantity types so that their usage has no impact on the runtime performance. The simulation results clearly show the restrictions of the TDF MoC concerning the simulation of block diagram models of physical systems. The fast execution of data flow models is achieved by avoiding any control of the numerical error and the absence of convergence iterations and dynamic time stepping features. A more physical-aware MoC is needed, which offers a similar flexibility in describing the abstract behaviour of modules, but causes less numerical problems during simulation.

5.7. SCAX Bond Graph (BG) MoC for the SystemC AMS extensions

This section describes how the bond graph formalism presented in Section 5.3.2 can be implemented as a new MoC for the SystemC AMS extensions. As the OSCI SystemC AMS extensions 1.0 LRM [131] does not standardise the synchronisation layer, the implementation of the BG MoC in the SCAX library is based on the internal details of the Fraunhofer SystemC-AMS PoC library implementation [53].

5.7.1. Requirements for the BG MoC

The new MoC needs to fulfil several requirements to successfully conserve the unique properties of the bond graph formalism and to integrate them into the modular architecture of the SystemC AMS extensions. To be able to describe a wide range of the CT behaviour found in heterogeneous multiphysical systems, it needs to support at the same time the description of conservative and non-conservative behaviour, i.e., mixed bond graph / block diagram models. The structure of these models, which is usually given as a graphical schema, needs to be expressed in form of a textual description. The natural approach is to provide it in the form of a *netlist*, which instantiates predefined primitives and links their ports via directed signals and bonds, similar to the LSF and ELN models already supported by the SystemC AMS extensions. To manage complexity, it must be possible to hierarchically describe the structure of the bond graph / block diagram models (i.e., the concept of *word bond graphs* [94]) with the help of user-defined (structural) macro models, which can be instantiated in parallel to the primitives in a structural description.

The BG MoC needs to offer the user a set of common bond graph and block diagram primitives (cf. to Tables 5.2 and A.4 on pages 89 and 154, respectively, for examples). These primitives need to be as flexible in their parameterisation as the block diagram primitives implemented for the TDF MoC in Section 5.6. This especially concerns the possibility to specify the quantity type or physical domain used by ports, directed signals, and bonds. The primitives shall use quantity parameters and quantity variables to describe their internal behaviour. These two requirements enable the checking of the coherency of the assembled model by means of static dimensional analysis during the compilation of the model. The user shall be able to add his own user-defined bond graph and block diagram primitives, which can use all capabilities of the BG MoC that are used by the predefined primitives. The effort in writing such user-defined primitives shall be comparable to writing a primitive TDF module.

The BG MoC needs to be able to analyse the structure of the assembled bond graph / block diagram model during elaboration. This includes the automatic assignment of causality to the bonds of the model based on constraints assigned by the primitives connected to the bond. The BG MoC shall report problems detected during the causality assignment, such as causality constraint conflicts and arbitrarily assigned causality. The latter hints to the presence of algebraic loops in the model. Based

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

on the assigned causality, the BG MoC shall execute the model in a procedural way (similar to the TDF MoC) by executing in the right order processing methods doing the necessary computations, which are registered by the bond graph and block diagram primitives based on the assigned causality. Contrary to the TDF MoC, the BG MoC shall use a dynamic time step and check for convergence of the solution after each iteration to decide whether it is possible to advance the solution by a time step. This means that the BG MoC shall not set up a global DAE system for the model, which is solved by a numerical solver. Instead algebraic constraints imposed by the presence of algebraic loops shall be solved locally. In a first step, fixed-point iterations [26, 162] can be used for this purpose. Future versions of the BG MoC may implement more elaborate numerical root-finding algorithms (e.g., Newton-Raphson or secant methods) to guarantee convergence of the solution under less stringent conditions than the fixed-point iterations algorithm imposes⁶. The goal of these measures is to optimise the simulation performance by avoiding the solution of a complex equation system and to use instead the assigned causalities to order the individual equations describing the CT behaviour for a sequential execution. This requires the BG MoC to perform a dependency analysis between the CT variables and the computations done with them in order to extract the computational structure. The BG MoC shall report any problems, e.g.:

- Variables which are not driven by any or are driven by multiple processing methods.
- Algebraic loops formed by variables and processing methods.

This will allow the user to not only gain insight into the physical structure of the modelled system but also into the computational structure of its model. This can reveal structural problems of the model and too ideal modelling assumptions with respect to the physical reality.

The BG MoC shall support the tracing of bonds, signals, and ports as well as the tracing of individual variables of bonds, ports, and modules (i.e., their internal state) during transient simulation by supporting the `sca_trace()` mechanism of the SystemC AMS extensions.

The new MoC shall integrate itself naturally into the architecture of the SystemC AMS extensions. The added language constructs shall follow the naming scheme used by SystemC and the AMS extensions. Concepts, which can be found similarly in other MoCs, shall not expose “surprising” behaviour to a user, who is familiar with the other MoCs. The BG MoC shall implement synchronisation with the other MoCs via the synchronisation layer of the AMS extensions. Following the example of the MoCs defined by the AMS extensions, this means that the BG MoC has to define converter ports and converter modules towards the TDF and DE MoCs. Thus, Continuous-Time (CT) non-conservative and conservative behavioural descriptions can be combined with Discrete-Time (DT) and Discrete Event (DE) behavioural descriptions to a model of a heterogeneous multiphysical system. Ideally this would also include the support for switching events in the bond graph and block diagram models. This requires the support of controlled junctions to define hybrid bond graphs (Section 5.3.2, Beers et al. [16]). However, the caused dynamic causality changes during the transient simulation would complicate the synchronisation semantics and the implementation of a first prototype of the BG MoC. Therefore, switching is not yet addressed in this work.

5.7.2. Architecture of the BG MoC

Figure 5.12 shows how the requirements for the BG MoC are translated into a general architecture that is integrated with the Fraunhofer SystemC-AMS PoC implementation. The BG MoC is implemented in

⁶The treatment of algebraic loops and resulting convergence problems in the context of the BG MoC is topic of Section 5.7.5.3.

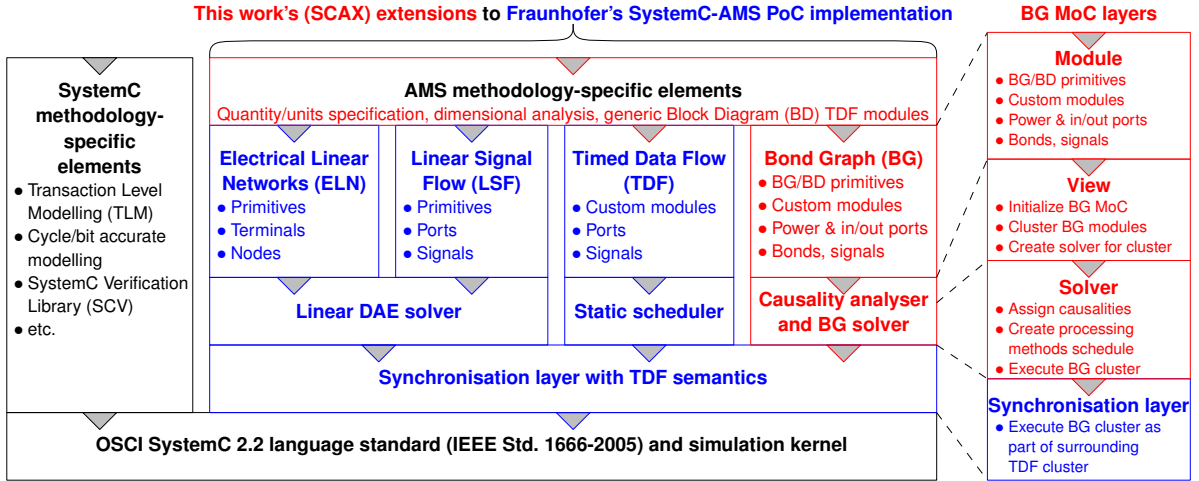


Figure 5.12.: Integration of the SCAX library into the architecture of Fraunhofer's SystemC-AMS PoC implementation. The SCAX library primarily uses the standardised APIs for SystemC [79] and its AMS extensions [131]. To integrate its BG MoC, it uses the internal synchronisation layer API of the Fraunhofer's SystemC-AMS PoC implementation, which is based on the TDF semantics. This allows the different Discrete-Time (DT) and Continuous-Time (CT) MoCs to synchronise every TDF module time step.

three architectural layers on top of the synchronisation layer of SystemC-AMS, which is also used by the ELN, LSF, and TDF MoCs of SystemC-AMS itself.

The *module layer* provides the user interface to the BG MoC. A new *BG module* class is the base for all predefined and user-defined bond graph and block diagram primitives. Each module can have instances of *directed signal input or output ports* as well as *power ports* for bonds. Additionally, the module may contain *converter ports* (a.k.a. *synchronisation ports*) to other MoCs (i.e., TDF and DE). Two channel classes implement the concept of a *directed signal* and of a *power bond*. The directed signals and bonds can be used to interconnect the BG modules inside an ordinary SystemC model to a macro model. The latter can contain instances of the directed signals and power ports to facilitate the hierarchical description of a mixed bond graph / block diagram model. The physical domain or quantity type of bonds, directed signals, and ports can be specified during instantiation. For the user's convenience, the BG MoC already proposes a *set of predefined BG modules* of common bond graph and block diagram primitives. The module layer is described in detail in Section 5.7.3.

The *view layer* handles the initialisation of the BG MoC. It is automatically constructed by the first instantiated BG module and every new BG module is registered with the view layer. During elaboration, after SystemC has carried out the hierarchy flattening and finished the binding of the ports to the channels, SystemC-AMS activates the view layer to do MoC-specific elaboration steps. To this end, the view layer clusters together all BG modules interconnected by directed signals or bonds. During this process, it also collects information about the converter ports instantiated by the BG modules. For each cluster, the view layer creates a BG solver instance and assigns the modules, bonds, directed signals, and converter ports forming the cluster to this BG solver. Its final responsibility is to elaborate the synchronisation time step of each BG solver with the other MoCs based on the module time steps assigned by the user to the BG module instances.

The *solver layer* is formed by the BG solver instances created during elaboration by the view layer.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

After the view layers have finished their elaboration for the different MoCs, SystemC-AMS activates the initialisation routine of each solver. The BG solver then registers itself with the synchronisation layer via the global SystemC-AMS simulation context as the responsible solver object for the BG cluster. At this occasion, it specifies a *cluster initialisation method* to be called before the start of simulation and a *cluster processing method* to be called during transient simulation to advance the solution of the BG cluster from one synchronisation point in time to the next one. Once SystemC-AMS activates the cluster initialisation method before the start of simulation, the BG solver asks each BG module of the cluster to specify its causality constraints on the bonds bound to its bond ports. Based on these constraints, the BG solver assigns, according to the priority of the set constraints, the causality to the bonds of the cluster and checks the result. Then, the BG solver asks each of its associated BG modules to register their processing methods for the transient simulation based on the causality assigned to its ports and to specify for each processing method its external input and output variables, which are accessed via the BG module's ports. Based on this information, the BG solver creates a dependency graph of the variables and processing methods to derive a schedule for the ordered execution of the processing methods during transient simulation. Finally, the BG solver calls each BG module to initialise their internal state. During transient simulation, the BG solver's registered cluster processing method is activated by the synchronisation layer once for every synchronisation point in time. The BG solver then advances the solution of the cluster from the last synchronisation point in time to the new one doing one or more variable time steps. For each time step, it executes the registered processing methods of the BG modules in the determined order. After each iteration, it checks for convergence of the solution. In case of non-convergence, it repeats the execution of the processing methods to perform fixed-point iterations until the solution can be accepted. If the number of fixed-point iterations crosses a certain threshold, the solver rejects the current time step and tries with a smaller one. Once the BG solver has advanced the cluster solution till the current synchronisation point in time, it returns control back to the synchronisation layer. The elaboration and simulation of BG models is described in detail in Section 5.7.4.

The *synchronisation layer* coordinates the parallel execution of the different continuous-time MoCs and offers to them a common interface to interact with each other and with the DE simulation kernel of SystemC. The OSCI SystemC AMS extensions standard [131] leaves the synchronisation layer implementation-defined. In the Fraunhofer SystemC-AMS PoC implementation of the standard, this layer uses the TDF semantics to synchronise the execution of the different Continuous-Time (CT) MoCs. Clusters of connected modules belonging to non-TDF CT MoCs (e.g., the standard ELN and LSF MoCs, but also the new BG MoC) are encapsulated into a synchronisation object, which contains the solver for the associated MoC. This synchronisation object has the same semantics and properties as an ordinary TDF module⁷ and is incorporated into the surrounding cluster of interconnected TDF modules and executed by the synchronisation layer during transient simulation. As a consequence, TDF ports and TDF converter ports can be used inside the modules of non-TDF CT MoCs to serve as synchronisation ports towards the TDF and DE MoCs.

5.7.3. Module Layer of the BG MoC

The module layer of the BG MoC is implemented by the classes shown in Figure 5.13, which are all defined in the namespace `scax_bg`. These classes form the base for the implementation of all predefined and user-defined bond graph and block diagram primitives presented later in this section. In the following,

⁷In fact, each TDF module is mapped on exactly one synchronisation object during elaboration.

the functionality of these classes will be presented as well as how physical domains are defined.

5.7.3.1. Overview on the Classes implementing the Module Layer

The first important element is the class `scax_variable<T>`, which implements a CT variable of data type `T`. This means that it actually stores two values: the current value and the last value at which the solution converged at the last time step. Like the later discussed signal, bond, and module classes, it implements the `scax_simulation_cycle_if`, which allows the BG solver to accept (`update()`) its current value as a converged solution (and thus, making the current value also the last converged one) or to reject (`reset()`) it (making the last converged value the new current value). Due to this property, objects of type `scax_variable<T>` are used to represent the internal state of directed signals, bonds, and modules. The latter implement the `scax_simulation_cycle_if` by forwarding calls to it to their internal `scax_variable<T>` instances. The class `scax_variable<T>` supports all mathematical operations of `T` and can be transparently used in calculations involving the data type `T`.

The class `scax_traceable_cref<T>` provides in the public interface of a class a named constant reference to a private member variable of data type `T`. This reference can be traced with the help of the `sca_trace()` mechanism of the SystemC AMS extensions. Thus, the internal state of a class is accessible for read-only tracing during simulation and, at the same time, efficiently protected against external manipulations bypassing the member functions of the class.

Two channel classes are provided for the BG MoC to represent directed signals and power bonds. The class `scax_signal<T>` implements a directed CT signal of data type `T`. It implements the `scax_signal_inout_if<T>`, which can be bound to the directed signal input port `scax_in<T>` and output port `scax_out<T>`. For the input port exists a multiport variant `scax_multiin<T, Attribute>`, to which an arbitrary number of signals can be bound⁸, e.g., to implement a summing module with arbitrary input number. For each bound signal, `scax_multiin<T, Attribute>` allows to store the value of an arbitrary `Attribute` type. This can be, e.g., a flag that the bound input signal shall be negated. The directed signal gives via its interface access to its current value and the last value, at which the solution converged for the previous time step. The directed signal and related ports can be traced with the help of the `sca_trace()` mechanism.

The class `scax_bond<Domain>` implements a bidirectional power bond for a physical domain specified through the template parameter `Domain`. The BG MoC provides several predefined physical domains in the namespace `scax_bg::domain` in form of template classes, e.g., `electrical<>`, `translational<>`, `rotational<>`, and `hydraulic<>`. The `Value` and `System` arguments of these physical domain classes allow to specify the value type (default: **double**) and the system of units (default: SI), respectively, to be used in the calculations. Based on this physical domain type, the quantity types of the `effort_` and `flow_` variables of the bond are derived with the help of the `scax_domain_traits<Domain>` class. The definition of physical domains and automatic derivation of related quantity types for calculations will be discussed in more detail in Section 5.7.3.2. The power bond class provides an interface to specify the causality constraints (in form of a `scax_causality_constraint_type` enum value) on the bond and assign a causality (in form of a `scax_causality_type` enum value) to it. The corresponding power ports (a.k.a. bond ports) implemented by the class `scax_port<Domain>` cannot be bound directly to the bond, but have to be bound either to its `head` or `tail`, which implement the `scax_bond_if<Domain>`. The bond side, to

⁸Multiports are normally supported by `sc_core::sc_port<IF>`, but, unfortunately, the SystemC AMS extensions 1.0 standard defines that AMS ports derived from `sca_core::sca_port<T>` need to be bound to exactly one channel [131, clause 3.2.4].

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

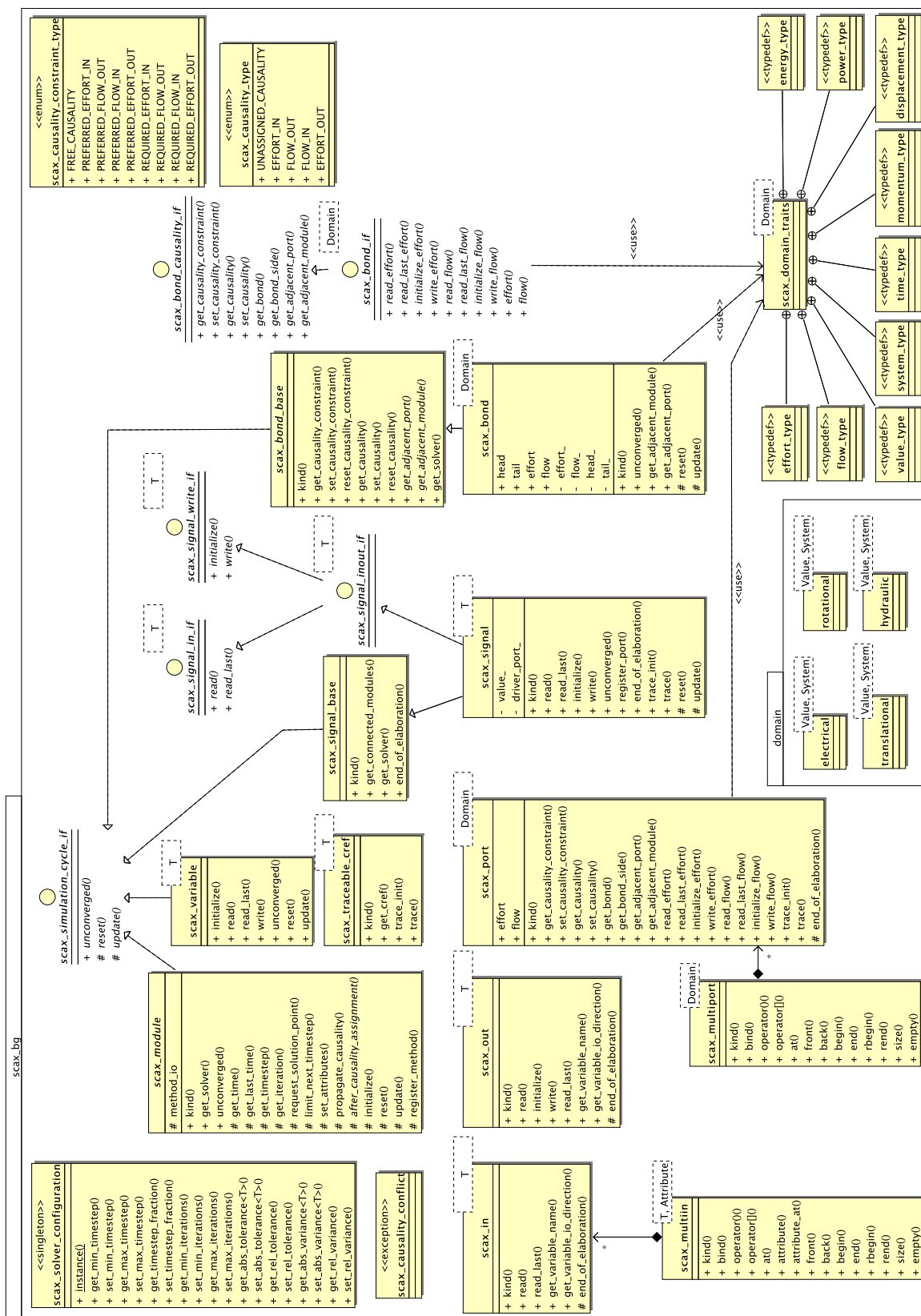


Figure 5.13.: Diagram of the classes constituting the module layer of SCAX's Bond Graph (BG) MoC for Fraunhofer's SystemC-AHS.

which a bond is bound, determines the sign of the effort and flow variables as well as the causality seen from the port, as the bond represents a directed energy flow. Again, a multiport variant of the power port is available in form of the `scax_multiport<Domain>` class. The power bond and power ports as well as the individual effort and flow variables support the tracing via the `sca_trace()` mechanism. The internal structure of the `scax_bond<Domain>` class is illustrated in Figure 5.14 as part of a simple BG model illustrating the creation, elaboration, and simulation phases of BG models using the BG MoC.

The class `scax_module` is the base class of all primitive BG modules. It provides the user access to the current time, the point in time when the solution last converged, the current time step, and the fixed-point iteration count. This information is provided by the BG solver, to which any BG module is automatically registered. A BG module can instantiate as part of its interface ports of type `scax_in<T>`, `scax_out<T>`, `scax_port<Domain>`, `sca_tdf::sca_in<T>`, `sca_tdf::sca_out<T>`, `sca_tdf::sca_de::sca_in<T>`, `sca_tdf::sca_de::sca_out<T>`. Its internal state is represented by variables of type `scax_variable<T>`. The internal state can be made accessible for tracing via public instances of type `scax_traceable_cref<T>`. The class `scax_module` defines the interface for the BG solver to elaborate and simulate a BG module. This is done in the form of several virtual member functions, which need to be overloaded by deriving BG modules to implement its behaviour. These member functions (a.k.a. callbacks) are called by the BG solver at distinct points during the elaboration and the simulation cycle. The concept is very similar to the elaboration and simulation callbacks of TDF modules (Section 5.2.1)—only additional callbacks are defined to cope with the more complex elaboration and simulation cycle. First, the `set_attributes()` member function is called during elaboration by the BG solver. In this member function, the BG solver needs to specify the causality constraints on its power ports (of type `scax_port<Domain>`). During causality assignment, the BG solver calls the `propagate_causality()` member function each time, it assigned a causality to one of the power ports of the BG module. The BG module needs to overload this member function to implement the causal constraints between its power ports. It checks the assigned causalities against its constraints and, based on the results, sets causalities on other power ports. After causality assignment, the member function `after_causality_assignment()` is called by the BG solver. This member function needs to be overloaded by *every* BG module to register one or more of its processing member functions based on the assigned causalities. For each registered processing member functions, the external input and output variables need to be specified, which are accessed via the module's ports. Being able to register more than one processing member function per BG module is the biggest difference with respect to the TDF MoC, which only supports a single processing member function per TDF module. The `initialize()` member function is called just before the start of the transient simulation to give the BG module the possibility to initialise its internal state and write the initial values of its output variables to its ports. During simulation, the registered processing member functions are called for each solution iteration. They implement the transient behaviour of the BG module in a very similar way to the `processing()` member functions of TDF modules. At the end of each fixed-point iteration, the `unconverged()` member function is called by the BG solver to determine if the internal state of the BG module did converge. Based on this information, the BG solver may continue with fixed-point iterations or accept or reject the current the solution by respectively calling the `update()` or `reset()` member functions of the BG module, which forward this call to its internal state variables of type `scax_variable<T>`. A BG module has the possibility to request the BG solver to insert an additional solution point between the last point in time, at which the solution converged, and the current point in time via the `request_solution_point()` member function. Similarly, it can ask the BG solver to limit the next solution time step to the value passed to the `limit_next_timestep()` member function.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Finally, the `scax_solver_configuration` class implements a singleton object, which stores all configuration settings of the BG solver. The user can gain access to it in the test bench using its static `instance()` member function. Then, he can configure the different settings related to the variable time stepping, the fixed-point iterations, and the convergence criteria: type-independent relative and type-dependent absolute tolerances⁹ and variances¹⁰.

Based on the presented classes, the BG MoC offers a library of pre-defined non-conservative block diagram and conservative bond graph primitives. They are summarised in the Tables A.5 and A.6 of Appendix A, respectively. Their implementation will be discussed in Section 5.7.3.3. The user can define in the same way his own primitive BG modules. An example is given in Section 5.7.5.1.

5.7.3.2. Definition of Physical Domains

The physical domain in the BG MoC is primarily defined by the *dimension* of the effort and flow quantity types. The user might also want to specify the *value type* (default: **double**) and the *system of units* (default: SI) to be used by the quantity types. For each physical domain, this information can be implemented in a straightforward way as a templatised **struct**. For, example the electrical physical domain is defined as follows:

```
1  template<typename Value = double,
2      typename System = si::system>
3  struct electrical {
4      typedef quantity<unit<electric_potential_dimension, System>,
5          Value> effort_type;
6      typedef quantity<unit<current_dimension, System>,
7          Value> flow_type;
8  };
```

The template arguments of the struct specify the value type and system of units. These two arguments are combined with the corresponding dimension via **typedefs** to the effort and flow type in the struct body. Just for comparison, this definition is equivalent to the definition of a **nature** in VHDL-AMS:

```
1  subtype VOLTAGE is REAL tolerance "DEFAULT_VOLTAGE";
2  subtype CURRENT is REAL tolerance "DEFAULT_CURRENT";
3  nature ELECTRICAL is VOLTAGE across
4      CURRENT through ELECTRICAL_REF reference;
```

The physical domain definition for the BG MoC is just missing the specification of a reference node. This concept of generalised networks has no equivalence in the bond graph formalism.

Using the described technique, The BG MoC defines in the namespace `scax_bg::domain` several basic physical domains: `electrical<>`, `translational<>`, `rotational<>`, and `hydraulic<>`. The user can define in the same way his own physical domains.

All other quantity types (e.g., to represent time, momentum, displacement, power, and energy) needed for the description of the energy conserving behaviour can be derived from the effort and flow types using the typedefs defined in the traits class¹¹ `scax_bg::scax_domain_traits<Domain>` (Listing 5.13). All generic bond graph primitives implemented by the BG MoC rely only on the types defined by this traits class. Thus, they become independent of the exact definition of the type describing the physical

⁹ Allowed relative/absolute difference between the current value and previous value of a variable during fixed-point iterations for the current point in time.

¹⁰ Allowed relative/absolute difference between the current value of a variable and its last accepted value from the previous time step.

¹¹ See footnote 2 on page 92 for an explanation of the traits technique.

Listing 5.13: Physical domain traits class.

```

1  template<typename Domain>
2  class scax_domain_traits {
3  public:
4      // Effort and flow type defined in the Domain struct.
5      typedef typename Domain::effort_type effort_type;
6      typedef typename Domain::flow_type flow_type;
7      // Type of the value part of the quantities.
8      typedef typename effort_type::value_type value_type;
9      // System of units of the quantities.
10     typedef typename effort_type::unit_type::system_type system_type;
11     // Time type with the same value and system type as the effort and flow types.
12     typedef typename quantity<unit<time_dimension, system_type>,
13                             value_type> time_type;
14     // Generalised momentum and displacement types.
15     typedef typename multiply_typeof_helper<
16         effort_type, time_type>::type momentum_type;
17     typedef typename multiply_typeof_helper<
18         flow_type, time_type>::type displacement_type;
19     // Power and energy types.
20     typedef typename multiply_typeof_helper<
21         effort_type, flow_type>::type power_type;
22     typedef typename multiply_typeof_helper<
23         power_type, time_type>::type energy_type;
24     // Static assertions checking the consistency of the type definitions
25     // (same underlying value type, time type, and system of units).
26     // ...
27 };

```

domain. The traits class implementation assumes that Boost.Units quantity types are used to annotate the measure and thus keep the link to the physical domain. If a value-only type shall be used, a specialisation, for this traits class, needs to be provided with appropriate typedefs. SCAX provides these specialisations for **float**, **double**, **long double**, and `std::complex<T>` for users, who absolutely want to avoid the additional safety net provided by the dimensional analysis in favour of reduced compile times. However, their usage is *strongly discouraged* in the context of bond graph modelling, as it becomes impossible to distinguish the physical domains and to assert the bond graph properties such as energy conservation (cf. to Figures 5.5b and 5.5c on page 85).

5.7.3.3. Implementation of Non-Conservative and Conservative BG Modules

Based on the classes of the module layer presented in Section 5.7.3.1, BG modules with non-conservative and conservative behaviour can be defined.

As an example how to implement a module with non-conservative behaviour, Listing 5.14 shows the relevant code extracts from the implementation of the trapezoidal integrator block diagram primitive, which is part of the `scax_bond_graph` library and is defined in the namespace `scax_bg`. The integrator module is derived from the `scax_module` base class like any other BG module. In the beginning of the module definition, two **typedefs** derive with the help of the `scax_data_type_traits<T>` and `boost::units::multiply_typeof_helper<...>` traits the time data type and, based on it, the output data type. The port declaration of the input and output port uses the directed signal port classes

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

of the BG MoC. The line “`SCAX_BG_HAS_METHOD(scax_integ_trapez);`” specifies that the class `scax_integ_trapez` will register processing member functions. A similar `SCAX_BG_HAS_METHOD()` statement with the current class name as argument needs to be added to every class body defining a BG module. It contains some definition, which will be later used by the macro `SCAX_BG_METHOD()` to register with a convenient syntax the processing member function¹². The constructor names the module’s port and initialises the internal state variable. Since the integrator block has a purely non-conservative behaviour and no power ports, it does not need to overload the `set_attributes()` and `propagate_causality()` member functions. It just needs to register processing member functions in the `after_causality_assignment()` member functions. This is done with the help of the `SCAX_BG_METHOD()` macro, to which the name of the processing member function is passed¹³. Once a processing member function is registered, its external input and output variables accessed via the module’s ports have to be specified. This is done with the help of the data member `method_io` of class `scax_module`. An input variable is specified with **operator**<< and an output variable is specified with **operator**>>¹⁴. To understand the syntax, just imagine the `method_io` to be a placeholder for a block diagram primitive. Then, “<<” and “>>” connect its input and output signals, respectively. The behaviour of the integrator is implemented in two processing member functions. Member function `integrate_in()` just reads from the port `in` and updates the internal state of the integrator using the trapezoidal rule. The member function `write_state_to_out()` does for the first iteration an initial estimation of the new state value by doing a forward Euler step and then writes the integrator state to the port `out`. This implements Heun’s method [162], a common predictor-corrector method, which tries to improve with each fixed-point iteration the result of the integration for the current point in time. In the dependency graph of the processing member functions and their input and output variables, these two processing member functions act as sink and source, respectively. Therefore, the insertion of an integrator into a signal path breaks an eventual algebraic loop. The `initialize()` member function initialises the output signal with the initial value of the integrator’s state variable before the start of transient simulation. The `unconverged()`, `update()`, and `reset()` member functions just forward the call to the respective member functions of the internal state variable.

The implementation of a BG module implementing the conservative behaviour of a bond graph primitive follows the same structure. As an example, Listing 5.15 shows the relevant code extracts of the modulated 2-port transformer primitive with TDF input port defined by the `scax_bond_graph` library in the namespace `scax_bg`. In the beginning of the BG module definition are again **typedefs**. They derive with the help of the `scax_domain_traits<Domain>` class the data type of the effort and flow variables of the power ports `bp1` and `bp2`. Based on these types, the data type `parameter_type` of the TDF input port is derived, which will modulate the transformer modulus. The TDF port of type `sca_tdf::sca_in<T>` acts as a converter port to the TDF MoC in BG modules. Similarly, a TDF converter port of type `sca_tdf::sca_de::sca_in<T>` can act as a converter port to the DE MoC of SystemC inside a BG module. The same holds true for the corresponding output ports `sca_tdf::sca_out<T>` and `sca_tdf::sca_de::sca_out<T>`. With `SCAX_BG_HAS_METHOD()`, the BG module `scax_tdf_MTF` announces that it will register processing member functions. In the `set_attributes()` member function, the BG module specifies that it puts no particular causality constraints on its individual bond ports. However, the causalities of ports `bp1` and `bp2` are linked to each other as can be seen

¹²SystemC defines a similar macro `SC_HAS_PROCESS()`, which enables the convenient syntax for the registration of `SC_METHODs` and `SC_THREADS` [79, clause 5.2.8].

¹³This is very similar to the registration of a method process instance or thread process instance using `SC_METHOD()` or `SC_THREAD()` in SystemC [79, clause 5.2.9].

¹⁴SystemC uses a similar syntax to specify the static sensitivity of an unspawned process [79, clause 5.2.13].

Listing 5.14: Trapezoidal integrator block diagram module for the BG MoC.

```

1  template<typename T>
2  class scax_integ_trapez : public scax_module {
3  public:
4      // Typedefs for the port-related quantity types.
5      typedef T in_data_type;
6      typedef typename scax_util::scax_data_type_traits<T>::time_type time_type;
7      typedef typename boost::units::multiply_typeof_helper<T, time_type>::type
8          out_data_type;
9
10     scax_in<in_data_type> in;    // Input port.
11     scax_out<out_data_type> out; // Output port.
12
13     SCAX_BG_HAS_METHOD(scax_integ_trapez); // Module registers processing methods.
14     // Constructs a trapezoidal integrator of the passed name and initial value.
15     explicit scax_integ_trapez(const sc_core::sc_module_name& name,
16                               const out_data_type& initial_value = out_data_type())
17     : in("in"), out("out"), state_(initial_state) {}
18     // ...
19     // Returns the number of unconverged variables for the current iteration.
20     virtual int unconverged() const { return state_.unconverged(); }
21 protected:
22     // Registers the processing member functions.
23     virtual void after_causality_assignment() {
24         SCAX_BG_METHOD(integrate_in);
25         method_io << in; // Specify that integrate_in() reads from port "in".
26         SCAX_BG_METHOD(write_state_to_out);
27         method_io >> out; // Specify that write_state_to_out() writes to port "out".
28     }
29     // Writes the initial value to the output port.
30     virtual void initialize() { out.initialize(state_); }
31     // Use trapezoidal method to improve the initial Euler guess of state_.
32     virtual void integrate_in() {
33         using scax_util::sca_time_cast;
34         time_type h = sca_time_cast<time_type>(this->get_timestep());
35         state_ = state_.read_last() + 0.5 * h * (in.read_last() + in.read());
36     }
37     // Write the current state_ to the output.
38     virtual void write_state_to_out() {
39         using scax_util::sca_time_cast;
40         if (!get_iteration()) { // 1st iteration: Estimate state_ with Euler step.
41             time_type h = sca_time_cast<time_type>(this->get_timestep());
42             state_ = state_.read_last() + h * in.read_last();
43         }
44         out.write(state_);
45     }
46     // Accept the current value of the integrator's state variable.
47     virtual void update() { state_.update(); }
48     // Reject the current value of the integrator's state variable.
49     virtual void reset() { state_.reset(); }
50 private:
51     scax_variable<out_data_type> state_; // Internal state variable.
52 };

```

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

in Table 5.2 on page 89. The assignment of a causality to one of the transformer's power ports forces the opposite causality on the other power port. This inter-port causality constraint is implemented in the `propagate_causality()` member function. Based on the assigned causalities, the `after_causality_assignment()` member function registers two out of a set of four processing member functions with the help of the `SCAX_BG_METHOD()` macro and specifies its inputs and outputs with `method_io`. One processing member function scales the effort variable read from the port with effort-in causality and writes the result to the effort variable of the other power port. The other processing member function does the same for the flow variable. The scale factor is determined by the transformer modulus read from the TDF port `in`. This selective processing member function registration after the causality assignment allows BG modules to optimise their internal calculations according to the assigned causalities. This means that bond graphs are virtually transformed by the BG MoC into an equivalent signal flow model during elaboration.

5.7.4. Elaboration and Simulation Cycle of the Bond Graph Model of Computation

Figure 5.14 summarises the steps for creating, elaborating, and simulating a BG model with the help of the BG MoC using a very simple example of just one effort source connected via a bond to a resistor. The bond graph is translated into a netlist, which uses the classes from the module layer (Section 5.7.3) to instantiate the BG model. After the control is handed over to the SystemC kernel using `sc_start()`, the model has to be elaborated by the BG MoC in several steps before its simulation. Section 5.7.2 outlined how the BG-MoC-specific part of the elaboration is handled by its view layer and solver layer, and how its solver layer executes the BG model for transient simulation under the control of the synchronisation layer of SystemC-AMS. Figure 5.15 shows the classes, which interact together on the view layer and solver layer to handle the elaboration and simulation. The next two subsections give insight into how the central classes `scax_bg::scax_view` and `scax_bg::scax_solver` perform the individual steps under the control of SystemC-AMS.

5.7.4.1. Elaboration of BG Models

After the construction of the module hierarchy representing the model in `sc_main()`, the control is handed over to the SystemC kernel with the call to `sc_start()`. It then performs the following steps related to the elaboration of a BG model¹⁵:

1. The SystemC kernel calls the `before_end_of_elaboration()` member function of all instances of `sc_module`, `sc_port`, `sc_export`, and `sc_prim_channel`.
2. It elaborates the ports and modules to flatten the hierarchy. All ports are bound to a channel after this phase.
3. It calls the `end_of_elaboration()` member function of all instances of `sc_module`, `sc_port`, `sc_export`, and `sc_prim_channel`. With the first executed `end_of_elaboration()` call-back executed of an AMS module, SystemC-AMS takes over the elaboration of all instantiated AMS modules:
 - a) The view layer of each AMS-MoC, for which corresponding AMS modules have been instantiated, are activated with a call to `sca_view_base::setup_equations()`, in

¹⁵The full elaboration and simulation semantics of SystemC and its AMS extensions are defined in IEEE CS [79, clause 4] and OSCI AMSWG [131, clauses 4.1.3, 4.2.3, 4.3.3, and 5].

Listing 5.15: Modulated 2-port transformer bond graph module with TDF input for the BG MoC.

```

1  template<typename Domain1, typename Domain2>
2  class scax_tdf_MTF : public scax_module {
3  public:
4      // Typedefs for the quantities related to the two bond ports bp1 and bp2...
5      typedef typename scax_domain_traits<Domain1>::effort_type bp1_effort_type;
6      typedef typename scax_domain_traits<Domain2>::effort_type bp2_effort_type;
7      // ...
8      // Linear parameter data type.
9      typedef typename boost::units::divide_typeof_helper<
10         bp1_effort_type, bp2_effort_type>::type parameter_type;
11      // ...
12      sca_tdf::sca_in<parameter_type> in; // Input port.
13      scax_port<Domain1> bp1;             // First bond port.
14      scax_port<Domain2> bp2;             // Second bond port.
15
16      SCAX_BG_HAS_METHOD(scax_tdf_MTF); // Module registers processing methods.
17      // Construct a modulated 2-port transformer with the passed name.
18      explicit scax_tdf_MTF(const sc_core::sc_module_name& nm)
19      : scax_module(nm), in("in"), bp1("bp1"), bp2("bp2") {} // ...
20  protected:
21      // Set causality constraints.
22      virtual void set_attributes() {
23          bp1.set_causality_constraint(FREE_CAUSALITY);
24          bp2.set_causality_constraint(FREE_CAUSALITY);
25      }
26      // Implement causality constraints between bond ports.
27      virtual bool propagate_causality() {
28          // Safe currently assigned causalities.
29          scax_causality_type bp1_causality = bp1.get_causality();
30          scax_causality_type bp2_causality = bp2.get_causality();
31          // Propagate assigned causalities from one bond port to the other.
32          if (bp1_causality != UNASSIGNED_CAUSALITY) {
33              bp2.set_causality(opposite_causality(bp1_causality)); }
34          if (bp2_causality != UNASSIGNED_CAUSALITY) { /* ... */ }
35          // Return true if at least one of the causalities changed.
36          if ((bp1.get_causality() != bp1_causality)
37              || (bp2.get_causality() != bp2_causality)) { return true; }
38          return false;
39      }
40      // Based on the assigned causalities, register the processing member functions.
41      virtual void after_causality_assignment() {
42          if (bp1.get_causality()==EFFORT_IN && bp2.get_causality()==EFFORT_OUT) {
43              SCAX_BG_METHOD(scale_e1_to_e2);
44              method_io << in << bp1.effort >> bp2.effort;
45              SCAX_BG_METHOD(scale_f2_to_f1);
46              method_io << in << bp2.flow >> bp1.flow;
47          } else if (bp1.get_causality()==FLOW_IN && bp2.get_causality()==FLOW_OUT) {
48              // ...
49          } else { SC_REPORT_ERROR("/SCAX/bond_graph", "Unexpected_causalities."); }
50      }
51      // Processing member functions.
52      void scale_e1_to_e2() { bp2.write_effort((-1.0/in.read())*bp1.read_effort()); }
53      void scale_f2_to_f1() { bp1.write_flow((-1.0/in.read())*bp2.read_flow()); }
54      // ... two more similar processing functions for the second causality case ...
55  };

```

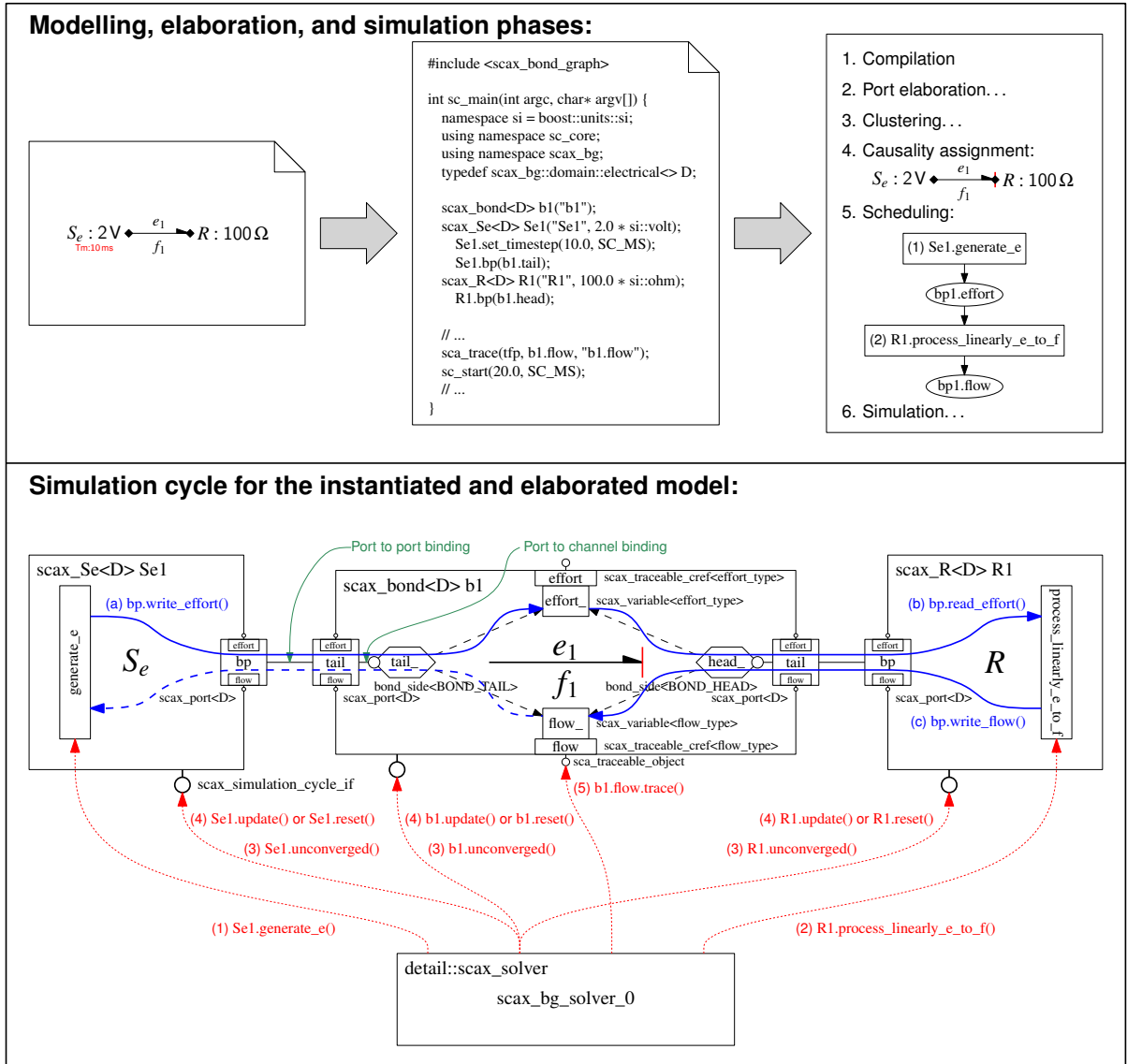


Figure 5.14.: Overview on the modelling, elaboration, and simulation phases for a bond graph model using SCAX's BG MoC for SystemC-AMS. The lower part shows the structure of the instantiated objects, which represent the bond graph model described in `sc_main()`. The **red arrows** denote, together with their labels, the order of member function calls performed by the BG solver to the individual objects of the model during one iteration. The **blue arrows** denote the order of access to the effort and flow variables of the bond from within the processing member functions of the effort source and resistor module instances.

5.7. SCAX Bond Graph (BG) MoC for the SystemC AMS extensions

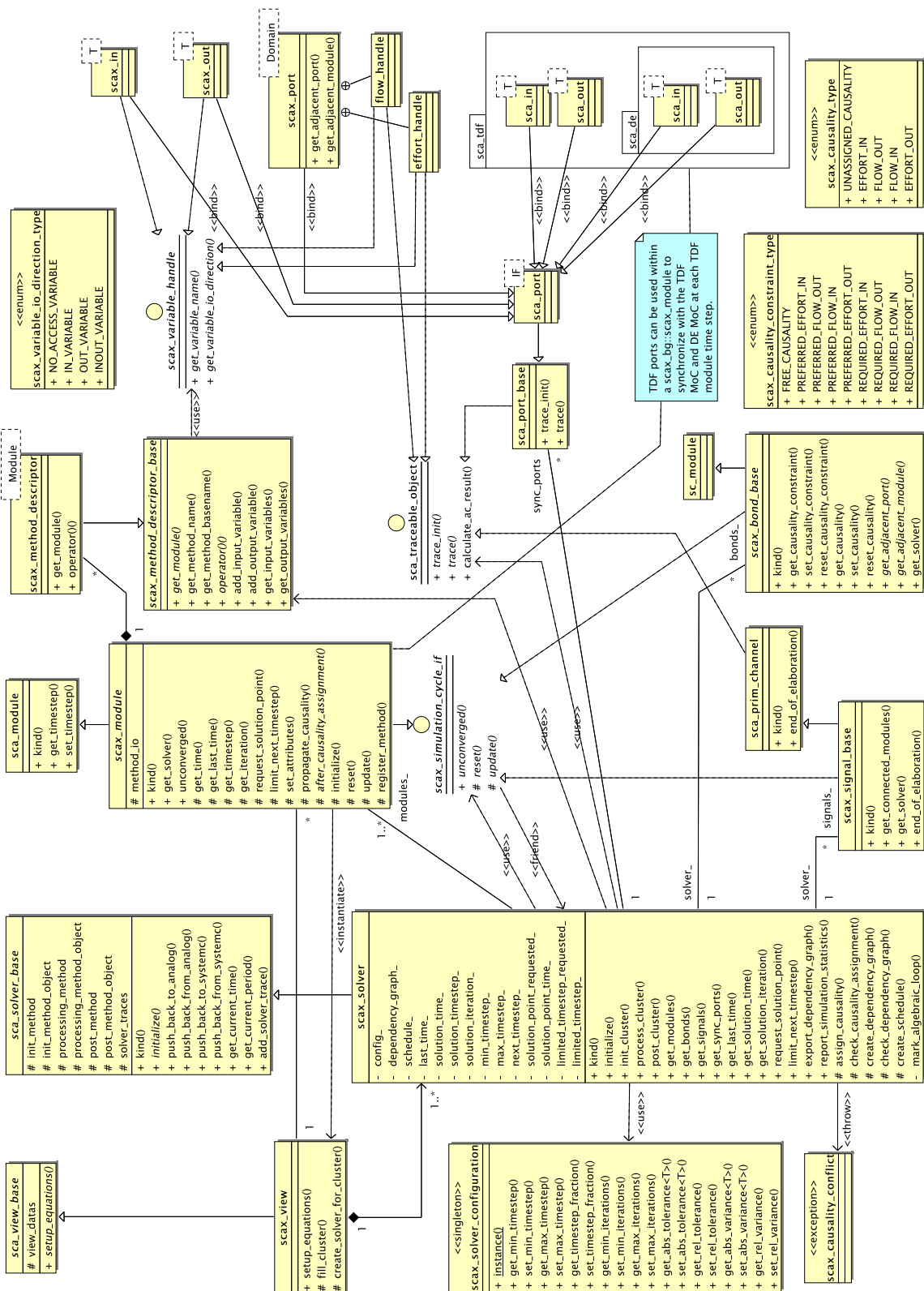


Figure 5.15.: Diagram of the classes related to the view layer and the solver layer of SCAX's BG MoC for Fraunhofer's SystemC-AMS.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

order to cluster connected AMS modules of the same kind and to set up the describing equation systems. The BG MoC implements its view layer with the class `scax_bg::scax_view`, which inherits from `sca_view_base`. It performs the following steps in `scax_bg::scax_view::setup_equations`:

- i. Based on its list of all instantiated BG modules, it starts from one BG module and traverses the module hierarchy with a depth first search [159] by following all directed (BG) signals and power bonds connected to its port to visit the adjacent modules. On its way through the module hierarchy, it records all:
 - Visited BG modules,
 - Followed directed signals and bonds,
 - Found converter ports of type `sca_tdf::sca_in<T>`, `sca_tdf::sca_out<T>`, `sca_tdf::sca_de::sca_in<T>`, `sca_tdf::sca_de::sca_out<T>` instantiated within a BG module.

These elements form together a BG cluster. The clustering process is repeated until all BG modules have been associated to a BG cluster.

- ii. For each BG cluster, a BG solver object of type `scax_bg::scax_solver` is instantiated and the control over the cluster is handed over to it. The solver is initialised by its constructor with some additional support coming from the view:
 - A. The constructor of the BG solver initialises its internal state and saves the information about the associated BG cluster.
 - B. The view checks if TDF synchronisation module time steps have been assigned to any BG module of the BG cluster. If that is the case, they must have all the same value, as from the perspective of the TDF-based synchronisation layer of SystemC-AMS, the BG MoC constitutes a single rate cluster. This value is set as TDF synchronisation time step of the BG solver. If no module time step was specified, SystemC-AMS will try to propagate a synchronisation time step to the solver from a possibly instantiated surrounding TDF cluster. SystemC-AMS enforces the existence of such a synchronisation time step and will abort the elaboration with an error if none has been assigned.
 - C. The view registers each converter port found, which is part of the BG cluster, with BG solver specifying its direction (input, output) and kind (from/to TDF MoC or from/to DE MoC).
- b) After the different views have terminated their clustering and constructed all solvers, SystemC-AMS calls `sca_solver_base::initialize()` on each solver to continue with the elaboration at the cluster level. For the BG MoC, this means the execution of the `initialize()` member function of each BG solver instance, which performs the following steps:
 - i. The BG solver registers itself and the associated converter ports with the synchronisation layer via the global SystemC-AMS simulation context. The synchronisation layer assumes the BG solver as owner of the converter ports and treats it in the following like an ordinary TDF module that fully encapsulates the BG cluster and has just the registered converter ports for communication with the outside world. This TDF module will be executed as part of the surrounding TDF cluster.

- ii. The BG solver registers the *cluster initialisation callback* (`init_cluster()`), *cluster processing callback* (`process_cluster()`), and *post-processing callbacks*¹⁶ (`post_cluster()`) with the synchronisation layer, which correspond semantically to the `initialize()`, `processing()`, and `end_of_simulation()` of a TDF module.
- c) SystemC-AMS proceeds by calling the `set_attributes()` member function of all instantiated TDF modules. The latter will then configure the TDF attributes of itself (module time step) and of its TDF (converter) ports (rate, delay, time step; cf. to Section 5.2.1).
- d) Based on the set TDF attributes, the synchronisation layer of SystemC-AMS creates the static schedule for the execution of each TDF cluster of connected TDF modules and encapsulated solver objects of other AMS MoCs, which will be used during transient simulation. Based on the port rates and assigned port and module time step, it calculates and assigns the missing port and module TDF time steps. At this moment, all TDF attributes are elaborated and the synchronisation (TDF) time steps are propagated throughout the TDF cluster.
- e) SystemC-AMS proceeds by calling the `initialize()` callback of each TDF module and registered *cluster initialisation callback* of each instantiated solver. This initialises the internal state of each module and solver for time $t = 0$ s. This will execute the `init_cluster()` member function of each `scax_bg::scax_solver` object, which does the following steps:
 - i. Call the `set_attributes()` member function of each BG module so that it specifies its causality constraints on the power ports.
 - ii. Assign causality to the bonds of the cluster using a generalised Sequential Causality Assignment Procedure (SCAP) adapted from Karnopp, Margolis, and Rosenberg [94] and implemented in `scax_solver::assign_causality()`:
 - A. Sort list of bonds with unassigned causality by priority of the causality constraints: first all bonds with required causality, then all bonds with preferred causality, and finally all bonds with free causality.
 - B. Process each bond in the list according to the priority:
 - Assign causality to the first bond in the list according to its causality constraint and move it to the end of the processed bond list. If this bond had no causality constraint assigned (free causality), then issue a warning that the bond graph contains an algebraic loop.
 - Activate the `propagate_causality()` member function on the two adjacent BG modules so that the BG modules verify their inter-port causality constraints and can assign additional causalities to connected bonds via their power ports. Bonds issue an error in form of a causality exception if the assigned causality does not matches a previously set required causality constraint. They issue a warning if the assigned causality does not match a preferred causality constraint. If this causes a causality exception, the causality assignment will be rolled back to the last causality assigned directly by the SCAP. This causality will then be reversed and the algorithm will activate again the `propagate_causality()` member function on the two adjacent BG modules to propagate the newly set

¹⁶The post-processing member functions are a relic from the old SystemC-AMS prototype [170]. They are not part of the SystemC AMS extensions 1.0 standard [131].

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

causality. If there is a causality exception thrown and no rollback point is left, the algorithm terminates with an error.

- Search the list of bonds, for bonds, which got a causality assigned in consequence of the calls to the `propagate_causality()` member function of BG modules, move them to the end of the processed bonds list. Activate in turn the `propagate_causality()` member function on the adjacent BG modules of each bond found. If there are no more bonds with assigned causality in the list, start again to assign causality to the first bond in the unprocessed bonds list.
- iii. Check the causality assignment, report statistics on the assigned causalities (taking into account the causality constraints), and report any problems (causality conflicts with assigned causality constraint, bond has no assigned causality). The check of the assigned causalities is implemented in `scax_solver::check_causality_assignment()`.
- iv. Call the `after_causality_assignment()` member function of each BG module so that they register their processing member functions and specify their external input/output variables accessed via the module's ports.
- v. Create a schedule for the ordered execution of the modules' registered processing member functions (cf. to `scax_solver::create_schedule()`). The implementation relies on the Boost.Graph library [159]:
 - A. Create dependency graph (directed graph) between processing member functions and their input/output variables (cf. to the member function `create_dependency_graph()` of class `scax_bg::scax_solver`):
 - Each variable and processing member function (identified by their unique hierarchical name) constitutes a vertex of the graph.
 - An input variable of a processing member function adds a directed edge from the variable to the processing member function.
 - An output variable of a processing member function adds a directed edge from the processing member function to the variable.
 - B. Check the dependency graph for any problems (cf. to the member function `check_dependency_graph()` of class `scax_bg::scax_solver`):
 - Report a warning if a registered processing member function has no input or output variables at all.
 - Report an error if any variable has no in-edge (no driver).
 - Report an error if any variable has more than one in-edge (multiple drivers) connecting it to multiple processing member functions.
 - C. Do a topological sort of the dependency graph to establish the schedule:
 - Do a *depth first search* on the graph and record the order of the *finished vertices* by the algorithm. Also mark the *back edges* encountered during the graph traversal, as they mark an *algebraic loop* with the adjacent vertices being the head and tail of the loop.
 - The *reverse order of the finished vertices*, which represent a processing member function, constitutes the schedule.

- D. Report the schedule.
- E. Report the breaking of algebraic loops at the back edges detected by the depth first search. Mark all edges belonging to the algebraic loop (intersection of the set of edges reachable from the loop head and the set of edges, from which the loop tail can be reached).
- vi. Call the `initialize()` member function of each BG module so that they can initialise their internal state and write initial values to the output variables of their ports.
- 4. The SystemC kernel finishes its part of the elaboration and proceeds to the transient simulation.

5.7.4.2. Simulation of BG Models

SystemC executes the following steps to perform the transient simulation:

1. The SystemC kernel calls for all instances of `sc_module`, `sc_port`, `sc_export`, and `sc_prim_channel` their `start_of_simulation()` member function.
2. The SystemC kernel starts its evaluation, update, notification cycle to execute all processes until the SystemC time $t_{SC} > t_{end}$:
 - a) SystemC-AMS evaluates each TDF cluster always at the first δ cycle of the synchronisation point in time with SystemC:
 - i. Process the TDF cluster schedule by executing the registered processing member function of each TDF module and AMS solver object ahead of the SystemC time. For this purpose, SystemC-AMS manages its own time (accessible via the `get_time()` member function of `sca_core::sca_module()`). The cluster execution will also activate the `process_cluster()` member function of any BG solver object that is part of the TDF cluster:
 - A. Get current solver time t_{solver} from synchronisation layer.
 - B. Get current solver period Δt_{solver} from synchronisation layer, for which the solution of the BG cluster needs to be advanced.
 - C. $t_{solver,end} := t_{solver} + \Delta t_{solver}$
 - D. Get minimum and maximum allowed time steps ($t_{step,min}$, $t_{step,max}$) from the `scax_solver_configuration` singleton object.
 - E. Last point in time the solution converged: $t_{last} := t_{solver}$
 - F. Current solution time: $t_{solution} := t_{solver}$
 - G. Next time step: $t_{next,step} := t_{step,max}$ or if limited time step $t_{step,limit}$ has been requested previously: $t_{next,step} := t_{step,limit}$
 - H. Get minimum and maximum number of allowed fixed-point iterations ($n_{iter,min}$, $n_{iter,max}$) from the `scax_solver_configuration` singleton object.
 - I. Get fraction $f_{t,step}$ for reducing the time step in case of non-convergence from the `scax_solver_configuration` singleton object.
 - J. Progress the solution of the BG cluster until the end of the solver period: While $t_{last} < t_{solver,end}$ do:

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

- Update the current point in time t_{solution} by either setting it to a requested solution point in time $t_{\text{requested}}$ or adding the next time step $t_{\text{next,step}}$ to t_{solution} . Limit t_{solution} to the end of the solver period $t_{\text{solver,end}}$.
- Solution time step $t_{\text{step,solution}} := t_{\text{solution}} - t_{\text{last}}$
- Try to converge a solution by doing fixed-point iterations.
For $i := 0$ to $n_{\text{iter,max}} - 1$ do:
 - Call each processing member function registered by the BG modules according to the schedule determined during the elaboration.
 - Sum up the number of unconverged variables of BG modules, power bonds, and directed (BG) signals (for each of them call their `unconverged()` member function and sum up the return value).
 - Break the iteration if there are no more unconverged variables.
- If an additional solution point has been requested or there are still unconverged variables and $t_{\text{step,solution}} > t_{\text{step,min}}$, then:
 - Reject the current solution by calling the `reset()` member function of all BG modules, bonds, and directed (BG) signals.
 - If the solution has been rejected due to unconverged variables, then reduce the next time step to either $t_{\text{step,next}} := \max(f_{t,\text{step}} \cdot t_{\text{step,next}}, t_{\text{step,min}})$ (no limited time step requested) or $t_{\text{step,next}} := \max(\min(t_{\text{step,limit}}, f_{t,\text{step}} \cdot t_{\text{step,next}}), t_{\text{step,min}})$ (limited time step requested).
 - If a solution point has been requested and lies beyond the next solution point calculated with the reduced time step, then ignore the request.
- Else:
 - If there are still unconverged variables, then the solution did not converge. Issue a warning and try to advance with the minimum configured time step.
 - Accept the current solution by calling the `update()` member function of all BG modules, bonds, and directed (BG) signals.
 - Activate the `trace()` member function of all traceable objects registered to the BG solver through the `sca_trace()` mechanism of SystemC-AMS.
 - Last converged solution: $t_{\text{last}} := t_{\text{solution}}$
 - Minimum time step: $t_{\text{step,min}} := \min(t_{\text{step,min}}, t_{\text{solver,end}} - t_{\text{last}})$
 - If the number of convergence iterations $i \leq n_{\text{iter,min}}$, then double the next time step $t_{\text{step,next}} := 2 \cdot t_{\text{step,next}}$
 - If a limited time step has been requested, then limit the next time step to it.
- ii. After one cluster schedule period, the SystemC-AMS synchronisation layer suspends to SystemC so that it can catch up with the time.
- b) SystemC scheduler runs until the next synchronisation point in time:
 - i. Evaluate phase: Select all runnable processes and resume their execution.
 - ii. Execute all pending updates from the previous evaluate phase.

- iii. If δ notifications, then do a new evaluate/update cycle.
- iv. If there are timed notifications, then advance the SystemC time t_{SC} to the next discrete event.
- c) The synchronisation layer of SystemC-AMS samples all DE inputs from SystemC read by the AMS modules through their converter ports.
- 3. The SystemC kernel calls for all instances of `sc_module`, `sc_port`, `sc_export`, and `sc_prim_channel` their `end_of_simulation()` member function. SystemC-AMS calls all post-processing member functions¹⁷ registered by its instantiated AMS modules and AMS solvers. The BG MoC uses this mechanism to output simulation statistics for each BG cluster.
- 4. The module hierarchy is destructed.

5.7.5. Application Examples

This section presents several application examples to demonstrate the usage of the BG MoC, to show its interaction with other MoCs, and to discuss its advantages and limitations.

5.7.5.1. Electromechanical Transducer with Linked Micromechanical Resonator

For the first application example, the electromechanical transducer example from Section 5.3 is revisited. Its block diagram model for the TDF MoC has been already presented in Section 5.6.2. Two models of the transducer have been developed for the new BG MoC (Figure 5.16). One is a straightforward reimplement of the block diagram model using, this time, the block diagram primitives from the `scax_bond_graph` library (Table A.5 on page 156) instead of the ones from the `scax_tdf` library (Table A.4 on page 154). As the source code of the resulting BG model is very similar (just different namespace prefixes and signal types) to the source code of the TDF model (Listings 5.11 and 5.12 on pages 106 and 108, respectively), it will not be presented. Much more interesting is the implementation of the BG model (Figure 5.16a), which uses the bond graph primitives of the `scax_bond_graph` library (Table A.6 on page 157).

The transducer example has one bond graph primitive, which is not available as a predefined primitive in `scax_bond_graph`. It is the 2-port C-field [94], which represents the electromechanical transducer itself. Its implementation, in form of a user-defined BG module `C2_transducer`, is shown in Listing 5.16 on page 133. The implementation closely follows the presented structure for BG modules, as presented in Section 5.7.3.3. The transducer's two ports `bp1` and `bp2` can be parameterised with the template parameters `Domain1` and `Domain2` to different physical domains. The 2-port C-field has two internal state variables `q1_` and `q2_`, which are traceable via the constant references `displacement1` and `displacement2`. Like the TDF model of the transducer (Listing 5.11), the `C2_transducer<Domain1, Domain2>` can be parameterised via its constructor upon instantiation with two transfer functions, which link the output efforts to the internal state variables ($e_1 = f_1(q_1, q_2)$ and $e_2 = f_2(q_1, q_2)$) described by the function wrappers `function_q1_q2_to_e1_` and `function_q1_q2_to_e2_`, respectively). Additionally, initial conditions can be specified. To reduce the complexity of the example, only a restricted form of the primitive has been implemented, which requires the optimum flow-in causality on its two power ports (cf. to `C2_transducer<>::set_attributes()`).

¹⁷See footnote 16 on page 125.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

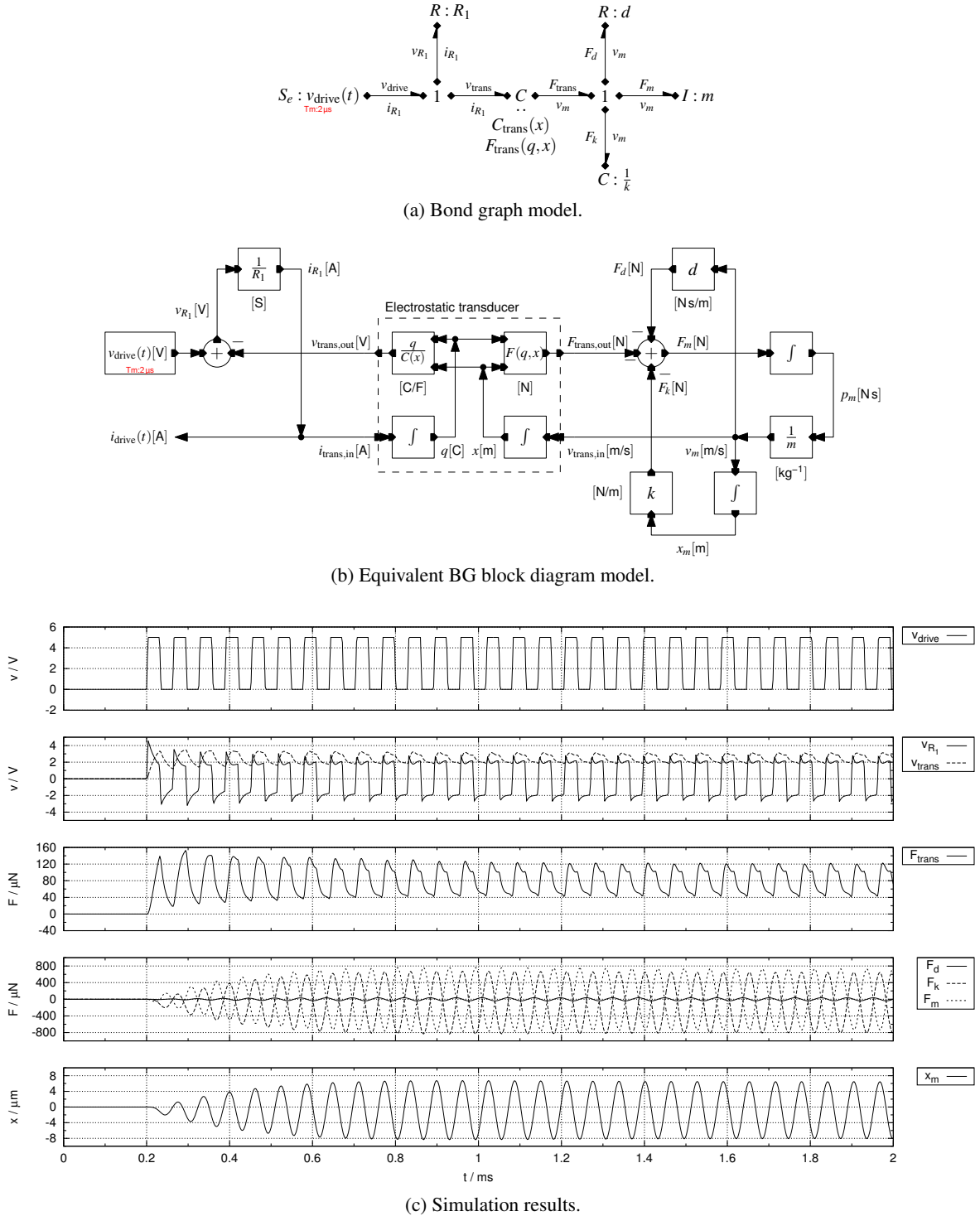


Figure 5.16.: Schematics and simulation results of the electromechanical transducer example using the BG MoC. Please refer to Table A.3 on page 153 for an explanation of the graphical convention used in the schematics. Both, the bond graph and equivalent block diagram model, yield the same behaviour and have no need to insert artificial delays. Due to the BG MoC's capability to perform convergence iterations, a module time step of $2\mu\text{s}$ instead of 10ns is sufficient to yield the same simulation results as the equivalent TDF model (Figure 5.11). The simulation performance is indicated in Table 5.4.

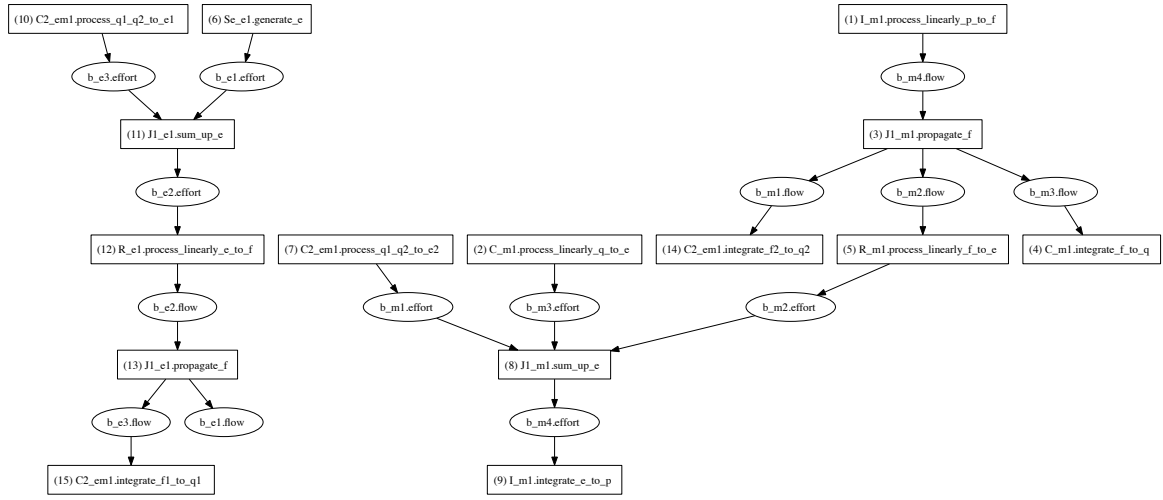
Supporting preferred causality constraints on both ports, like it is the case for the 1-port C primitive (`scax_bg: : scax_C<Domain>`, Table A.6 on page 157), would require the implementation of more processing member functions to cope with all four possible causality combinations on the two ports and, consequently, would require more transfer function parameters in the constructor. The implemented optimum flow-in causality for the two bond ports causes the integration of the flow variables to the displacement state variables. The effort variables are calculated from the displacements using the transfer functions $f_1(q_1, q_2)$ and $f_2(q_1, q_2)$. The description of this behaviour is distributed on four independent processing member functions `integrate_f1_to_q1()`, `integrate_f2_to_q2()`, `process_q1_q2_to_e1()`, and `process_q1_q2_to_e2()`. The two integrations are implemented again using Heun's method [162], which has been already used by the integrator block diagram primitive (Listing 5.14) discussed in Section 5.7.3.3. The processing member functions are registered in the `after_causality_assignment()` callback, together with their respective input/output variables.

The test bench instantiating the `C2_transducer<Domain1, Domain2>` class as part of the bond graph describing the electromechanical transducer with linked micromechanical resonator is shown in Listing 5.17 on page 136. The initialisation of all the model parameter constants in the beginning of the `sc_main()` function is not shown, as it is identical to the corresponding section of the TDF model shown in Listing 5.12 on page 108. Instead, the focus lies on the instantiation of the bond graph primitives and their interconnection with bonds. It can be seen that it is very similar to the structural descriptions of a regular SystemC model. The only major difference is that a power port cannot be directly bound to a bond, but has to be bound either to the *head* or *tail* of the bond. Before the start of simulation, the variables to be traced are registered via `sca_trace()` calls. In the code extract given in Listing 5.17, the effort variable of bond `b_e1` and the displacement variable of capacitor `C_m1` are traced. After the configuration of the TDF synchronisation time step ($2\ \mu\text{s}$) and the minimum simulation time step ($10\ \text{ns}$), the transient simulation is started for $2.4\ \text{ms}$.

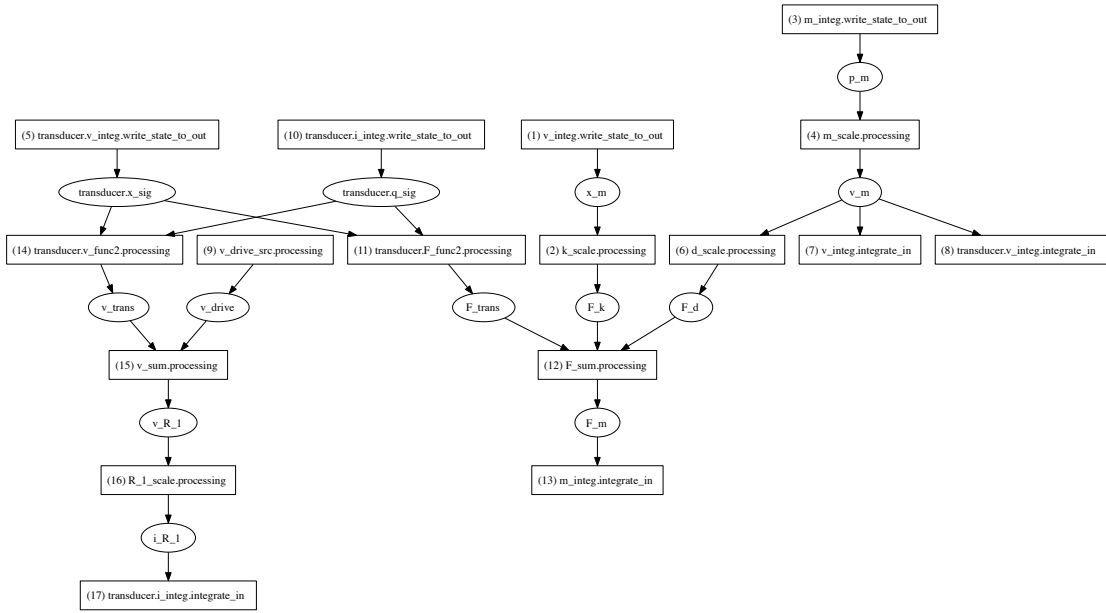
One distinct feature of the BG MoC, which is not present in any other MoC of SystemC-AMS, is the capability of its BG solver to export the dependency graph between the processing member functions and variables of its associated BG cluster, generated during elaboration, in the Graphviz format [50, 58]. This can be done after the simulation, by accessing the BG solver associated to a BG module and call its `export_dependency_graph()` member function. This graph visualises the computational structure of the model and can be helpful, e.g., to localise the origin of convergence problems due to algebraic loops (Section 5.7.5.3). The bond graph and block diagram models of the electromechanical transducer example do not have any algebraic loops, as can be seen in Figure 5.17. The graphs show that the bond graph variant registers only 15 processing methods, whereas the block diagram variant registers 17 processing methods. The difference simply comes from the fact that less bond graph primitives than block diagram primitives need to be instantiated in the respective models to describe the same behaviour. Despite the slightly different looking graphs, the computational structure of both model variants is equivalent. Just the hierarchical names differ and the dependency graph for the bond graph does not show the internal state variables of the C -field transducer (`q_sig` and `x_sig` in the `transducer` block diagram model, Figure 5.16b). This is due to the fact that only external input/output variables need to be specified for processing methods during their registration with the BG solver.

Figure 5.16c shows the simulation results of the implemented bond graph model. The pulsed input voltage v_{drive} excites the sinusoidal oscillation of the mechanical resonator at its natural resonance frequency. The common mode of the driving voltage shows up in the resulting electrostatic force and displacement of the resonator. The simulation results of the equivalent block diagram BG model are not given, as they are numerically identical due to the model's equivalent computational structure. The two models show the same behaviour as the equivalent block diagram TDF model presented in Section 5.6.2

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling



(a) Dependency graph of the bond graph model using the BG MoC.



(b) Dependency graph of the block diagram model using the BG MoC.

Figure 5.17.: Equivalent dependency graphs of the bond graph and block diagram models of the electromechanical transducer example generated by SCAX's BG MoC.

and the developed reference models in VHDL-AMS. This demonstrates the successful simulation of all major effects of this small system. Table 5.4 compares the compile and execution times for all the different VHDL-AMS, TDF, and BG model variants. It can be seen that the compilation of the SystemC models takes significantly longer than the compilation of the VHDL-AMS models due to the extensive usage of templates. Especially the usage of the Boost.Units library for the quantity types and compile-time dimensional analysis raises the effort for the compiler. However, the usage of quantity types instead of **double** has no negative impact on the simulation performance. With the same configured maximum time step t_{step} , all four SystemC models simulate faster than the two VHDL-AMS models. Contrary to the TDF model, which did not converge with a $t_{\text{step}} = 2 \mu\text{s}$, the BG models reliably converge with this time step like the VHDL-AMS models do. The BG MoC requires a slightly higher average number of convergence iterations per time step $\frac{\text{iterations}}{\text{steps}}$ compared to the VHDL-AMS models. Due to the better error control algorithm implemented in ADMS, more steps are calculated during the simulation of the VHDL-AMS than by the BG models, which explains the lower number of $\frac{\text{iterations}}{\text{steps}}$. In fact, the BG MoC currently implements variable time stepping for transient simulation (Section 5.7.4.2), but its primitives, which numerically integrate or differentiate quantities over time, do not yet implement an estimation of the numerical error and thus, do not yet use the provided time stepping control mechanism. Despite the similar number of accepted steps and average number of iterations per time step, the SystemC-AMS finishes the simulation of the BG models with a $t_{\text{step}} = 2 \mu\text{s}$ nearly 16 times faster than ADMS does for the VHDL-AMS models. However, it needs to be considered that ADMS is optimised for the simulation of much more complex models. Therefore, the loading of the more complex simulator and the more complex preprocessing of the models has a non-negligible impact on the time needed for small simulations like the presented example. It can be already observed in the presented results for a simulation with $t_{\text{step}} = 10 \text{ ns}$ that the simulation performance of BG models and VHDL-AMS models can get very similar if the equation system rapidly converges within a few iterations. Still, the considerable observed simulation performance advantage for BG models of multiphysical systems is very interesting—especially within the SystemC-AMS simulation framework. The TDF MoC is lacking any checking for the convergence of the solution, which makes it very sensitive to the choice of the simulation time step. The TDF MoC simulates about two times faster than the BG MoC at $t_{\text{step}} = 10 \text{ ns}$ because it blindly does only one iteration per time step instead of an average of two convergence iterations for the BG MoC. However, by increasing the simulation time step to $t_{\text{step}} = 2 \mu\text{s}$, which is sufficient to obtain the same precision, the BG model simulates nearly 50 times faster than the TDF model. Interestingly, the bond graph model shows a slight performance advantage over the equivalent block diagram model using the BG MoC, because it requires the registration of only 15 processing methods instead of 17 processing methods, respectively, to describe the same behaviour. So the usage of bond graph primitives instead of block diagram primitives can have advantages for the simulation performances. Additionally, as it has been already discussed in Section 5.7.3.3, the usage of bond graph primitives for models simplifies their reuse, since their causality is only fixed at elaboration time, and not already during the writing of the block diagram model, leaving more interconnection options with other physical component models. These results show that the BG MoC is much better suited for the simulation of nonlinear conservative/non-conservative CT behaviour than the TDF MoC in conjunction with the ELN and LSF MoCs.

Listing 5.16: Generic 2-port C-field transducer module.

```

1 #include "scax_bond_graph"
2 // ...
3 // Generic 2-port C-field transducer module with fixed effort out causality.
4 template<typename Domain1, typename Domain2>
```

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Table 5.4.: Comparison of the model compilation/execution performance of the electromechanical transducer example realised in different modelling languages and using different modelling capabilities. Each model variant underwent a transient analysis with the same parameters ($h_{\max} = t_{\text{step}}$, $t_{\text{stop}} = 2.4 \text{ ms}$, $\epsilon_{\text{ps}} = 10^{-6}$). The simulations were all done on a PC with 3 GHz Intel Pentium 4 Hyper-Threading CPU and 2 GiB RAM running Debian Lenny with a Linux 2.6.32 kernel. The TDF block diagram models did not converge for a $2 \mu\text{s}$ time step.

Language and simulator	Used features	$t_{\text{compile}}/\text{s}$	$t_{\text{execute}}/\text{s}$	$t_{\text{step}}/\text{ns}$	<i>steps</i>		$\frac{\text{iterations}}{\text{steps}}$
					accepted	rejected	
VHDL-AMS, Mentor Graphics Questa ADMS 2009.2a	free quantity	0.954	26.006	10	240 323	70	1.862
			3.990	2000	1526	70	3.170
	branch quantity	0.864	26.184	10	240 653	212	1.003
			3.650	2000	2198	320	3.374
SCAX 0.12.0, SystemC-AMS 1.0b2pre, SystemC 2.2.0 , g++ 4.4.5 (-O2)	scax_tdf, double	9.544	11.052	10 2000	240 001	0 incorrect results	1.000
	scax_tdf, quantity<U, V>	18.130	11.410	10 2000	240 001	0 incorrect results	1.000
	scax_bg, block diagram primitives	23.026	25.850	10	240 001	0	1.919
		23.008	0.280	2000	1201	0	4.974
	scax_bg, bond graph primitives	18.786	23.668	10	240 001	0	1.919
		18.756	0.230	2000	1201	0	4.974

5.7. SCAX Bond Graph (BG) MoC for the SystemC AMS extensions

```

5  class C2_transducer : public scax_bg::scax_module {
6  public:
7      // Typedefs for the quantities related to the two bond ports bp1 and bp2...
8      typedef typename scax_bg::scax_domain_traits<Domain1>::effort_type
9          bp1_effort_type;
10     typedef typename scax_bg::scax_domain_traits<Domain1>::displacement_type
11         bp1_displacement_type;
12     // ...
13     typedef typename scax_bg::scax_domain_traits<Domain2>::effort_type
14         bp2_effort_type;
15     // ...
16     // Function wrapper types.
17     typedef std::tr1::function<bp1_effort_type (bp1_displacement_type,
18         bp2_displacement_type)> function_q1_q2_to_e1_type;
19     typedef std::tr1::function<bp2_effort_type (bp1_displacement_type,
20         bp2_displacement_type)> function_q1_q2_to_e2_type;
21
22     scax_bg::scax_port<Domain1> bp1; // First bond port.
23     scax_bg::scax_port<Domain2> bp2; // Second bond port.
24
25     SCAX_BG_HAS_METHOD(C2_transducer); // Module registers processing methods.
26     // Construct the 2-port C-field transducer model.
27     C2_transducer(const sc_core::sc_module_name& nm,
28         function_q1_q2_to_e1_type function_q1_q2_to_e1,
29         function_q1_q2_to_e2_type function_q1_q2_to_e2,
30         bp1_displacement_type q1_0 = bp1_displacement_type(),
31         bp2_displacement_type q2_0 = bp2_displacement_type())
32     : scax_bg::scax_module(nm), bp1("bp1"), bp2("bp2"),
33         function_q1_q2_to_e1(function_q1_q2_to_e1),
34         function_q1_q2_to_e2(function_q1_q2_to_e2),
35         q1_(q1_0), q2_(q2_0),
36         displacement1("displacement1", q1_), displacement2("displacement2", q2_)
37     {}
38     // ...
39     // Returns the number of unconverged variables for the current iteration.
40     virtual int unconverged() const {
41         return q1_.unconverged() + q2_.unconverged(); }
42 protected:
43     // Set causality constraints.
44     void set_attributes() {
45         bp1.set_causality_constraint(scax_bg::REQUIRED_FLOW_IN);
46         bp2.set_causality_constraint(scax_bg::REQUIRED_FLOW_IN);
47     }
48
49     // Register the processing member functions depending on the assigned causality.
50     void after_causality_assignment() {
51         sc_assert(bp1.get_causality() == scax_bg::FLOW_IN);
52         sc_assert(bp2.get_causality() == scax_bg::FLOW_IN);
53         SCAX_BG_METHOD(integrate_f1_to_q1);
54         method_io << bp1.flow;
55         SCAX_BG_METHOD(integrate_f2_to_q2);
56         method_io << bp2.flow;
57         SCAX_BG_METHOD(process_q1_q2_to_e1);
58         method_io >> bp1.effort;
59         SCAX_BG_METHOD(process_q1_q2_to_e2);
60         method_io >> bp2.effort;

```

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

```

61 }
62
63 void update() { q1_.update(); q2_.update(); } // Accept the current solution.
64 void reset() { q1_.reset(); q2_.reset(); } // Reject the current solution.
65 private:
66 void process_q1_q2_to_e1() {
67     if (!get_iteration()) {
68         // For 1st iteration use an Euler step to estimate q1_ and q2_.
69         time_type h = scax_util::sca_time_cast<time_type>(get_timestep());
70         q1_ = q1_.read_last() + h * bp1.read_last_flow();
71         q2_ = q2_.read_last() + h * bp2.read_last_flow();
72     }
73     // For all succedent iterations, q1_ and q2_ have been already updated.
74     bp1.write_effort(function_q1_q2_to_e1(q1_, q2_));
75 }
76
77 void process_q1_q2_to_e2() { /* Similar to process_q1_q2_to_e1(). */ }
78
79 void integrate_f1_to_q1() {
80     using scax_util::sca_time_cast;
81     // Use trapezoidal method to improve the initial Euler guess of q1_.
82     time_type h = sca_time_cast<time_type>(get_timestep());
83     q1_ = q1_.read_last() + 0.5 * h * (bp1.read_last_flow() + bp1.read_flow());
84 }
85
86 void integrate_f2_to_q2() { /* Similar to integrate_f1_to_q1(). */ }
87
88 // Functions to describe the relation e1_ := f(q1_, q2_) and e2_ := f(q1_, q2_).
89 function_q1_q2_to_e1_type function_q1_q2_to_e1_;
90 function_q1_q2_to_e2_type function_q1_q2_to_e2_;
91
92 // Generalised displacement state variables for bp1 and bp2.
93 scax_bg::scax_variable<bp1_displacement_type> q1_;
94 scax_bg::scax_variable<bp2_displacement_type> q2_;
95 public:
96 // Traceable constant references to the state variables q1_ and q2_.
97 scax_bg::scax_traceable_cref<bp1_displacement_type> displacement1;
98 scax_bg::scax_traceable_cref<bp2_displacement_type> displacement2;
99 }; // class C2_transducer<Domain1, Domain2>

```

Listing 5.17: Test bench of the bond graph model of the electromechanical transducer example.

```

1 #include "scax_bond_graph" // ...
2 // Definition of C2_transducer<Domain1, Domain2>.
3 // ...
4 int sc_main(int argc, char* argv[]) {
5     // Using declarations and convenience typedefs.
6     using namespace scax_bg; using scax_bg::domain::electrical;
7     // ...
8     typedef divide_typeof_helper<si::force, si::velocity>::type viscous_damping;
9     // Electrical and mechanical component constants as well as stimuli constants.
10    const quantity<si::resistance> R_1 = 50.0e3 * si::ohm;
11    // ...
12    const sc_time t_sim(2.4, SC_MS); // Simulation time.
13    const sc_time t_step(2.0, SC_US); // TDF synchronisation time step.
14

```

```

15 // Bond graph definition.
16 scax_bond<electrical<> > b_e1("b_e1"), b_e2("b_e2"), b_e3("b_e3");
17 scax_bond<translational<> >
18   b_m1("b_m1"), b_m2("b_m2"), b_m3("b_m3"), b_m4("b_m4");
19 // Electrical driving circuit.
20 scax_Se<electrical<> >
21   Se_e1("Se_e1", scax_pulse<voltage_type>(
22     V_drive_0, V_drive_1, t_drive_delay,
23     t_drive_rise, t_drive_fall, t_drive_pulse, t_drive_period));
24   Se_e1.bp(b_e1.tail);
25 scax_J1<electrical<> > J1_e1("J1_e1");
26   J1_e1.bp(b_e1.head); J1_e1.bp(b_e2.tail); J1_e1.bp(b_e3.tail);
27 scax_R<electrical<> > R_e1("R_e1", R_1);
28   R_e1.bp(b_e2.head);
29 // Transducer.
30 C2_transducer<electrical<>, translational<> >
31   C2_em1("C2_em1", v_trans_func, F_trans_func, q_trans_0, x_m_0);
32   C2_em1.bp1(b_e3.head); C2_em1.bp2(b_m1.head);
33 // Mechanical resonator.
34 scax_J1<translational<> > J1_m1("J1_m1");
35   J1_m1.bp(b_m1.tail); J1_m1.bp(b_m2.tail);
36   J1_m1.bp(b_m3.tail); J1_m1.bp(b_m4.tail);
37 scax_R<translational<> > R_m1("R_m1", d);
38   R_m1.bp(b_m2.head);
39 scax_C<translational<> > C_m1("C_m1", 1.0 / k, x_m_0);
40   C_m1.bp(b_m3.head);
41 scax_I<translational<> > I_m1("I_m1", m, p_m_0);
42   I_m1.bp(b_m4.head);
43
44 // Tracing.
45 sca_trace_file *tfp = sca_create_tabular_trace_file("elmech_transducer_bg");
46 sca_trace(tfp, b_e1.effort, "v_drive"); // ...
47 sca_trace(tfp, C_m1.displacement, "x_m");
48 // Simulation.
49 Se_e1.set_timestep(t_step); // Always required TDF synchronisation time step.
50 // Optionally, set BG solver parameters like minimum solution time step, etc.
51 scax_solver_configuration::instance().set_min_timestep(10.0, SC_NS);
52 try { sc_start(t_sim); } catch (const exception& e) { cerr<<e.what()<<endl; }
53 sca_close_tabular_trace_file(tfp);
54 Se_e1.get_solver()->export_dependency_graph("elmech_transducer_bg.dot");
55 sc_stop(); // End simulation to print simulation statistics.
56 return sc_report_handler::get_count(SC_ERROR);
57 }

```

5.7.5.2. Interaction of the BG, TDF, and DE Models of Computation

After the discussion of writing and simulating pure BG models, this section focuses on the integration of BG models into a system using also other MoCs. The synchronised interaction of the BG MoC with other MoCs of SystemC and its AMS extensions has been an important requirement for the design and implementation of the BG MoC (Section 5.7.1). To this end, every BG module can instantiate the SystemC-AMS synchronisation ports of type `sca_tdf::sca_in<T>`, `sca_tdf::sca_out<T>`, `sca_tdf::sca_de::sca_in<T>`, `sca_tdf::sca_de::sca_out<T>` to interact with TDF modules or “classical” SystemC DE modules (Section 5.7.3.1). As each BG cluster is executed as part of the

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

surrounding TDF cluster, the synchronisation with the other MoCs is ensured (Section 5.7.4).

To facilitate the interconnection of BG, TDF, and DE models, several converter primitives are offered for the BG MoC by the `scax_bond_graph` library. For the block diagram formalism, these are source primitives, which are controlled by either a TDF or DE signal, and sink primitives, which convert a directed BG signal into a sampled TDF or DE signal (Table A.5 on page 156). Similarly for the bond graph formalism, dedicated modulated source, transformer, and gyrator primitives are offered as well as effort and flow detector primitives. Each of these primitives exist in three variants with either a directed BG signal port, TDF port or TDF to DE converter port (Table A.6 on page 157).

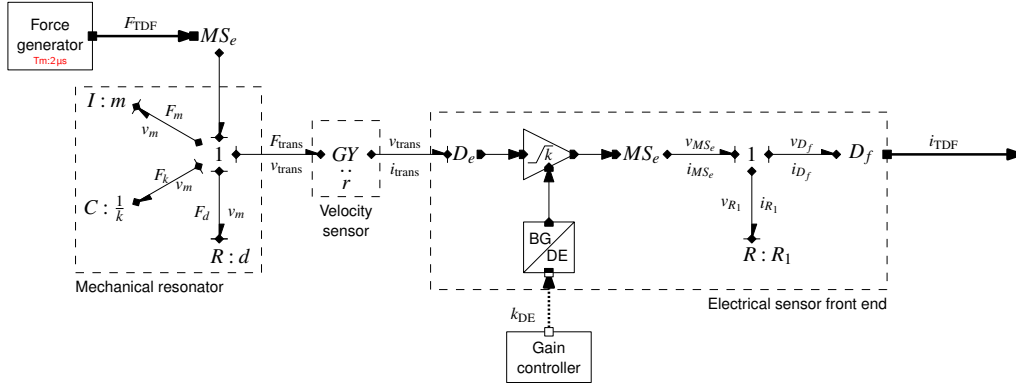
To demonstrate the resulting capabilities for writing heterogeneous system models involving different modelling formalisms and MoCs, a simple model of a sensor with electrical front end to measure the velocity of an externally excited mechanical resonator has been developed. Its schematic is shown in Figure 5.18a. The reader can refer to Table A.3 on page 153 for a summary on the used graphical convention to distinguish the different signal and port types. Bond graph and block diagram descriptions are combined in a single model. The mechanical resonator is described as a bond graph as well as is the electrical output circuit, which converts the output voltage of the amplifier into an output current. The velocity sensor is represented by a gyrator. Its electrical output signal (voltage v_{trans}) undergoes analogue signal processing modelled with a block diagram model. To this end, the voltage v_{trans} is converted into a directed signal and amplified by a gain block, controlled by the digital gain controller. The output saturates if it exceeds a given voltage range. The output of the gain block is connected to a modulated effort source, which converts the directed BG signal back into the energy conserving domain. The two bond graphs with the interconnecting block diagram form a single BG cluster. The mechanical resonator is excited by a modulated effort source, which is controlled by the TDF signal F_{TDF} coming from the force generator TDF module. The output current signal of the electrical front end is converted by a flow detector primitive to the TDF signal i_{TDF} . Just by replacing this detector with the right variant, it would be possible to obtain a sampled DE signal at the output.

This model can be assembled mostly from the primitives available in the `scax_tdf` and `scax_bond_graph` libraries (Tables A.5 and A.6 on pages 156 and 157, respectively). The only exception is the digital gain controller module, which needs to be hand-coded. This is not difficult, as it just steps the k_{DE} between different values and waits in between for a certain time. Therefore, no code examples are given, as they would not provide any significantly new insights into the writing of structural BG models.

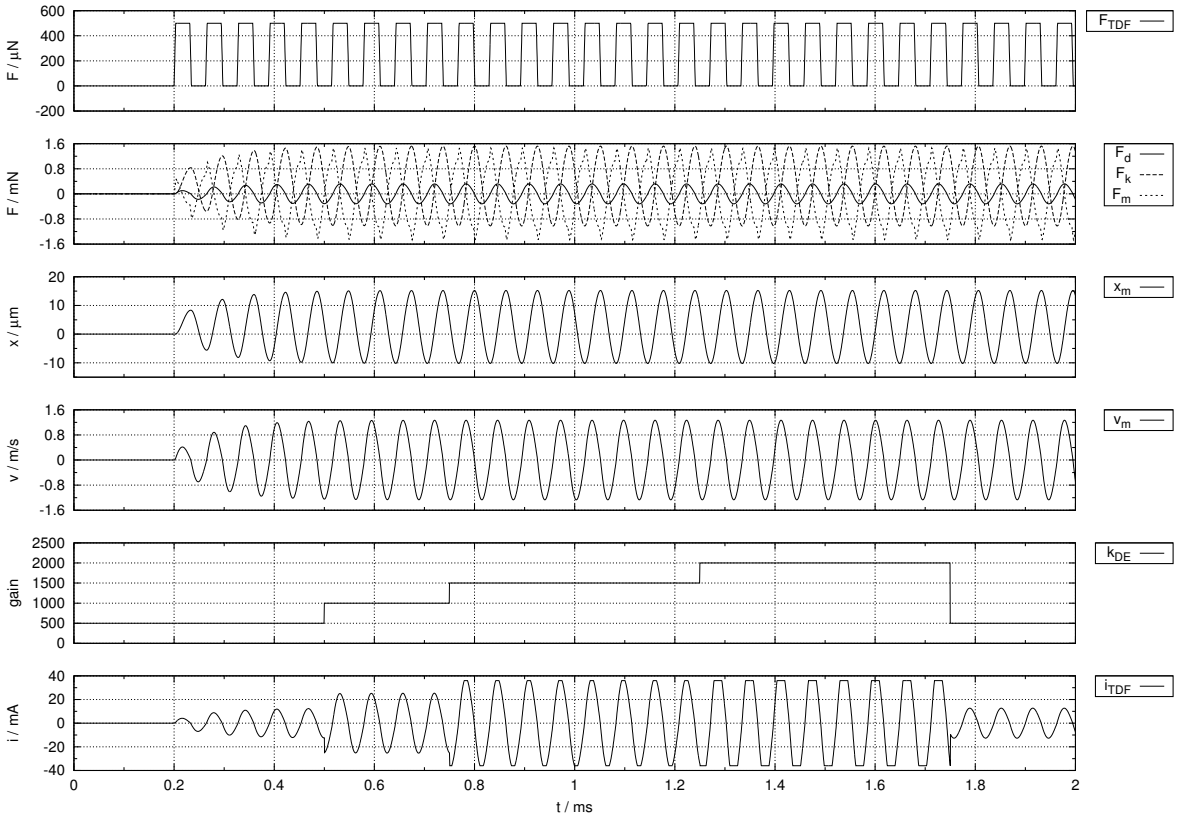
Figure 5.18b shows the obtained simulation results. For the mechanical resonator part, the behaviour is similar to the previous electromechanical transducer example (Section 5.7.5.1). The pulsed force signal F_{TDF} excites the oscillation of the resonator (displacement x_{m} , velocity v_{m}) at its natural frequency. The velocity sensor's output voltage is amplified and converted into a current i_{TDF} . On this output signal, the saturation of the amplifier can be observed when the gain control block sets a too high gain k_{DE} . This model demonstrates how tightly the BG, TDF, and DE MoCs can interact to simulate heterogeneous system models. However, one limitation of the current BG MoC implementation is the missing support for controlled junctions to describe hybrid bond graphs that show switching behaviour (Sections 5.3.2 and 5.7.1).

5.7.5.3. Treatment of Algebraic Loops

Algebraic loops can easily appear in physical models due to algebraic constraints between some of the model's variables. Their presence signifies that the modelled system cannot be described through a set of ODEs, but through a set of DAEs. This complicates the procedural solution of the equation system by



(a) Schematic of the sensor model using TDF, DE, and BG modules.



(b) Simulation results.

Figure 5.18.: Simple model of a sensor with electrical front end to measure the velocity of an externally excited mechanical resonator. The resonator is excited by a driving force, which is generated by a TDF source. The voltage gain of the electrical sensor front-end is controlled by a digital control block modelled using the DE MoC of SystemC. The amplifier limits its output to the power supply range. The model demonstrates that bond graph and block diagram models handled by the BG MoC can tightly interact with DE and TDF modules.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

ordering the system's equations according to the dependencies between the variables so that the source nodes of the yielded dependency graph are representing only those variables, which values are known at the solution point in time (i.e., state variables and variables, which depend through a known function only from the time t). All other variables are calculated through pure assignment from calculations with known variables. An algebraic loop is formed once a variable x is at the same time source and sink in the dependency graph (it is adjacent to a back edge of the directed graph) so that a constraint $x = f(x)$ needs to be fulfilled, where $f(x)$ is defined by the path through the dependency graph, which closes the loop. In a procedural solution, this requires fixed-point iterations, which are not guaranteed to converge. The Banach fixed-point theorem needs to be fulfilled [162], which requires $f(x)$ to fulfil the Lipschitz condition to guarantee convergence:

$$|f(x_2) - f(x_1)| \leq L \cdot |x_2 - x_1| \quad \text{with} \quad L < 1 \quad \text{for all} \quad x_1, x_2 \in \mathbb{R} \quad (5.6)$$

If this condition is not fulfilled, a numerical root-finding algorithm needs to be used, like the Newton-Raphson method or the secant method.

The BG MoC currently does not implement such a root-finding algorithm. It relies solely on fixed-point iterations for convergence, which makes it sensitive to the kind of models simulated. For example, models involving resistive networks often have convergence problems due to algebraic loops. Two very simple electrical examples are given in Figures 5.19 and 5.20. The bond graph model in Figure 5.19b does converge with the indicated (automatically assigned) causality, when using the resistance values $R_{11} = 1 \text{ k}\Omega$ and $R_{12} = 10 \text{ k}\Omega$. However, just by switching the resistance values, the solution does not converge. Similarly, the bond graph in Figure 5.20b does neither converge with the indicated (automatically assigned) causality, when using the resistance values $R_{31} = R_{33} = 1 \text{ k}\Omega$ and $R_{32} = R_{34} = 10 \text{ k}\Omega$, nor when using the resistance values $R_{31} = R_{33} = 10 \text{ k}\Omega$ and $R_{32} = R_{34} = 1 \text{ k}\Omega$. The resonator models presented in Sections 5.7.5.1 and 5.7.5.2 showed good convergence, because they do not contain any algebraic loops.

To overcome convergence problems, several approaches are possible. One possibility is to review the modelling assumptions to see which energy storing parasitics were neglected and to progressively add them until the causality can be completed without algebraic loops [94]. This increases of course the order of the ODE and will usually lead to a slow-down in simulation—especially if the equation system gets stiff with differences in its characteristic time constants of several orders of magnitudes. Another possibility is to analyse the bond graph and simplify it for simulation [94]. Connected junctions of the same type can be joined without changing the dynamic behaviour. Linear elements of the same type (R , C , or I), which are connected to the same junction or linked through a transformer or gyrator element, can be merged in a way, which is not changing the dynamic behaviour. This can lead to a reduction in the number of states/non-states and number of remaining algebraic loops. The BG MoC can, unfortunately, not help with these simplifications, as it does not have knowledge about the exact behaviour of each BG primitive required for the implementation of such simplification rules. This choice has been deliberate, because support for automatic simplification of bond graphs would require to significantly restrict the set of modelling primitives available to the user—in the simplest case to just primitives with linear dynamic behaviour. However, the BG MoC offers the user much more flexibility by giving him the possibility to write his own bond graph and block diagram primitives.

To help the user to diagnose convergence problems and solve them with the approaches just described, the BG MoC implements some features, which are not commonly found in other dynamic simulators. The BG MoC already detects the presence of algebraic loops during causality assignment (Section 5.7.4.1). When the implemented Sequential Causality Assignment Procedure (SCAP) does not terminate after

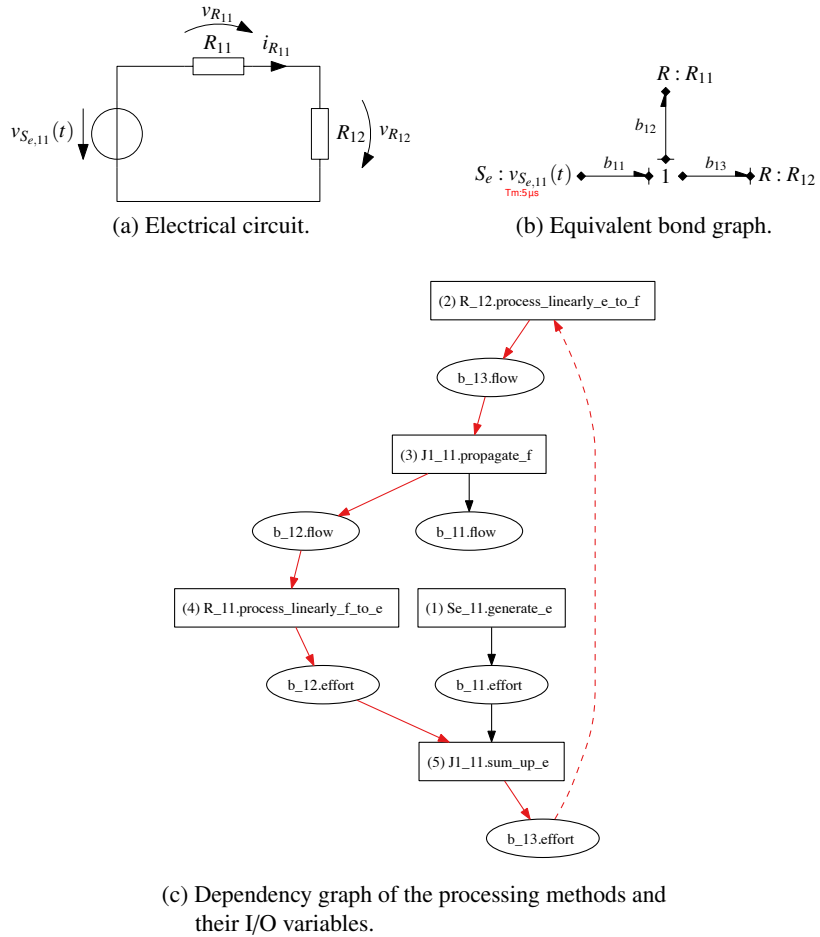
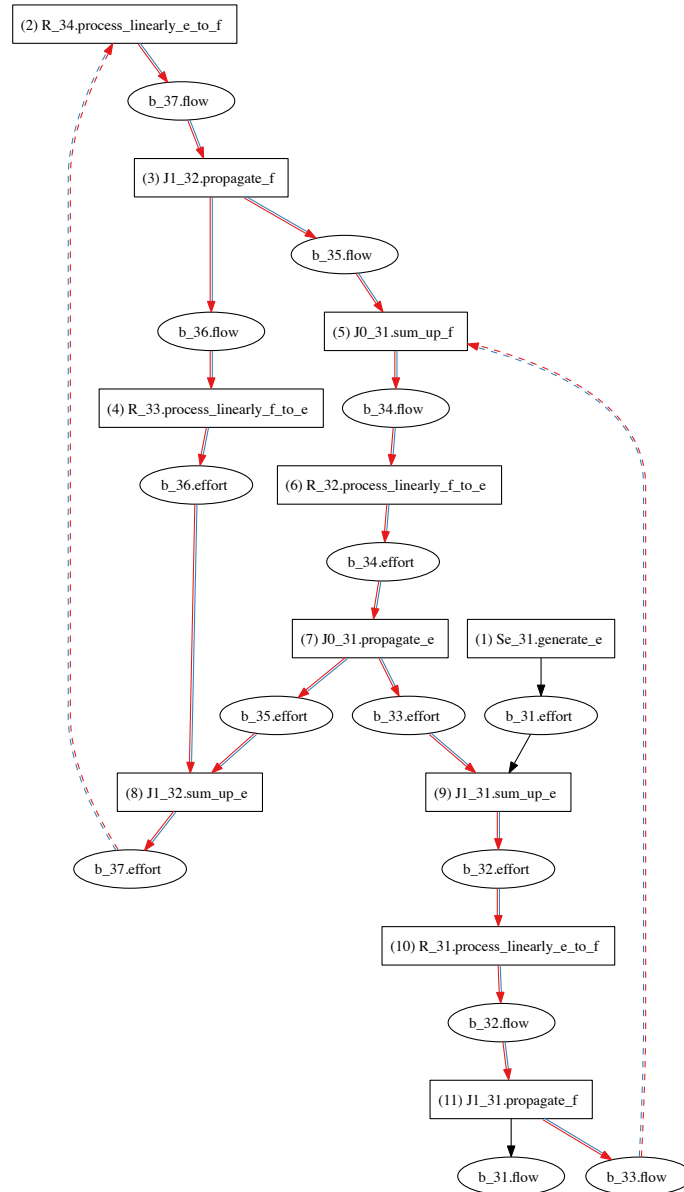
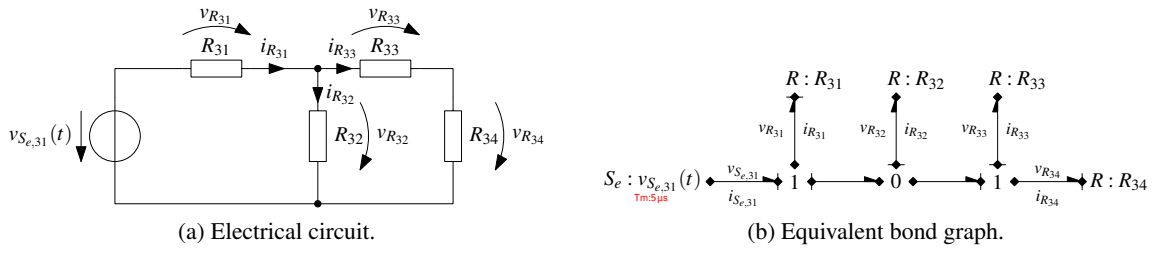


Figure 5.19.: Simple electrical example yielding a bond graph containing an algebraic loop: series connection of two resistors connected to a voltage source.

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling



(c) Dependency graph of the processing methods and their I/O variables.

Figure 5.20.: Simple electrical example yielding a bond graph with multiple algebraic loops: series and parallel connection of four resistors connected to a voltage source.

assigning causality to each bond with required or preferred causality constraints, it issues a warning that the model contains algebraic loops due to the unconstrained assignment of a causality to a bond. For example, it issues the report shown in Listing 5.18 after the automatic causality assignment to the bond graph shown in Figure 5.20b. Once the BG solver performs the dependency analysis of the registered processing member functions and their input/output variables, it can issue more precise warnings, which are shown in Listing 5.19.

As described in Section 5.7.5.1, the user can export the dependency graph generated by the BG solver into the Graphviz format to further diagnose the problem by getting insight into the computational structure of the model. Figure 5.20c depicts the dependency graph generated from the bond graph shown in Figure 5.20b. It can be seen that the BG solver marks the back edges, which close the algebraic loop with a dashed line, as these edges are not considered during the scheduling. The BG solver also marks with a distinct colour the edges, which are part of the respective algebraic loop, as they can be reached from the loop head (vertex of which the back edge is an in edge) and, at the same time, the *loop tail* (vertex of which the back edge is an out edge) is reachable from these edges. In the example given, two loops are marked, which are overlapping each other's "influence area". The dependency graph contains other important information: the hierarchical name of each registered processing member function and input/output variable as well as the order, in which the processing member functions will be executed during each convergence iteration. This information can help the user to find suitable places to add missing energy storing parasitics to enhance the convergence of the model or to find a starting point for simplifying the BG model with the same goal. A future version of the BG MoC may implement a root-finding algorithm locally into the bonds and signals, which can be selectively activated by the BG solver for the variables adjacent to a back edge in the dependency graph to help with the convergence of the model.

5.8. Conclusions and Outlook

This chapter presented the new modelling capabilities introduced with the SCAX library by the author of this Ph.D. thesis for the recently standardised OSCI SystemC AMS extensions. They enable the formal description and efficient simulation of heterogeneous multiphysical systems at high levels of abstraction. To this end, the chapter showed the strong relevance of dimensional analysis to ensure consistent model equations and proper assembly of models for multiphysical systems. It presented the capabilities of the OSCI SystemC AMS extensions as a C++-based multi-MoC simulation framework for the efficient HW/SW co-design of heterogeneous systems with a current focus on electrical AMS SoCs for communication and signal processing applications. In the scope of the presented Ph.D. thesis work, the author actively contributed to their recent standardisation as member of the OSCI AMS Working Group. The current limitations of the SystemC AMS extensions concerning the modelling of digitally assisted analogue and multiphysical components of heterogeneous SoCs were recognised and systematically addressed. Two suitable modelling formalisms were identified to describe their conservative and non-conservative behaviour at high levels of abstractions: bond graphs and block diagrams, respectively. The generic nature of their modelling primitives over "classical" domain-specific modelling primitives make them more universally applicable. However, their usage requires a more precise specification of the model component interfaces and their calculations, to not lose the link to the physical domain, and a more careful assembly of the structural system model, to avoid interconnection errors especially at physical domain boundaries. This can be achieved through the systematic usage of physical quantity types instead of purely computational data types (e.g., **double**) for the variables, parameters, ports, and

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

Listing 5.18: Causality assignment results for the bond graph depicted in Figure 5.20b.

```

1 Info: /SCAX/bond_graph: Bond graph cluster 'scax_bg_solver_1' is constituted of:
2   8 modules (e.g., 'R_34')
3   7 bonds (e.g., 'b_37')
4
5 Info: /SCAX/bond_graph: Bond causality assignment priority order in cluster ↓
   'scax_bg_solver_1': 'b_31' (unassigned constraint), 'b_37' (unassigned ↓
   constraint), 'b_32' (unassigned constraint), 'b_34' (unassigned constraint), ↓
   'b_36' (unassigned constraint), 'b_35' (unassigned constraint), 'b_33' ↓
   (unassigned constraint).
6
7 Info: /SCAX/bond_graph: Assigned causality to 7/7 bonds of cluster ↓
   'scax_bg_solver_1' in 7 iterations.
8
9 Warning: /SCAX/bond_graph: Some causalities have been assigned arbitrarily to ↓
   bonds in cluster 'scax_bg_solver_1' without any imposed constraints. This is ↓
   a hint for the presence of algebraic loops in the model.
10 In file: (...)scax/src/scax/bond_graph/detail/scax_solver.cpp:801
11 In process: b_11.sca_implementation_0.cluster_process_1 @ 0 s
12
13 Info: /SCAX/bond_graph: Summary of the causality assignment to the 7 bonds in ↓
   cluster 'scax_bg_solver_1':
14
15 -----
16 Causality      Required      Preferred      Unpreferred      Free      Total
17 -----
18 Effort in  --\|              1              0              0              3              4
19 Flow in   |--\              0              0              0              3              3
20 -----
21 Total              1              0              0              6              7
22 -----

```

Listing 5.19: Dependency graph analysis results for the bond graph depicted in Figure 5.20b.

```

1 Info: /SCAX/bond_graph: Dependency graph for bond graph cluster ↓
   'scax_bg_solver_1' has 25 vertices (11 processing methods, 14 variables) and ↓
   27 edges.
2
3 Info: /SCAX/bond_graph: Schedule for bond graph cluster 'scax_bg_solver_1': ↓
   'Se_31.generate_e', 'R_34.process_linearly_e_to_f', 'J1_32.propagate_f', ↓
   'R_33.process_linearly_f_to_e', 'J0_31.sum_up_f', ↓
   'R_32.process_linearly_f_to_e', 'J0_31.propagate_e', 'J1_32.sum_up_e', ↓
   'J1_31.sum_up_e', 'R_31.process_linearly_e_to_f', 'J1_31.propagate_f'.
4
5 Warning: /SCAX/bond_graph: Algebraic loop broken between 'b_33.flow' and ↓
   'J0_31.sum_up_f' in bond graph cluster 'scax_bg_solver_1'.
6 In file: (...)scax/src/scax/bond_graph/detail/scax_solver.cpp:1242
7 In process: b_11.sca_implementation_0.cluster_process_1 @ 0 s
8
9 Warning: /SCAX/bond_graph: Algebraic loop broken between 'b_37.effort' and ↓
   'R_34.process_linearly_e_to_f' in bond graph cluster 'scax_bg_solver_1'.
10 In file: (...)scax/src/scax/bond_graph/detail/scax_solver.cpp:1242
11 In process: b_11.sca_implementation_0.cluster_process_1 @ 0 s

```

signals of their models. In this way, the physical dimension and measurement unit are associated in a machine-readable form to the values used in computations and communication, which allows the model compiler to perform automated consistency checks on the calculations and structural model assembly.

The presented implementation in form of the seamless integration of the Boost.Units library with the SystemC AMS extensions achieves this goal without imposing a simulation performance penalty. The declaration of quantity ports, signals, variables, and parameters as well as the definition of numeric quantity constants with their measurement unit is compact, conforming to the C++ syntax, and close to the mathematical notation. The C++ compiler is performing the dimensional analysis for the mentioned consistency checks as part of the static type checking phase and optimises away all computational overhead for the model execution, which increases the compile time. The internal complexity of the quantity types is only exposed to the user in case of compiler errors reporting incoherent computations and wrong assignments involving incompatible quantity types. To address the resulting usability problem, the `bufilt` utility has been developed to simplify the compiler messages. It is able to localise, parse, and replace any occurrences of the complex unit template type, which encodes all aspects of a measurement unit (dimension, system of units) for the quantity type, by a compact human-readable format. This facilitates the localisation and correction of many problems before the first execution of the model. Thus, it gives the designer/programmer more time to test the important behavioural aspects of his model.

To conserve the genericity of bond graph and block diagram primitives despite the usage of the new quantity types, their interface and behaviour needs to be parameterisable in a flexible but well-defined way that allows to express constraints between the quantity types. This has been achieved by applying generic and functional programming techniques to the model writing. These techniques have been demonstrated through the implementation of a library of flexible block diagram primitives for the TDF MoC of the SystemC AMS extensions. It has been successfully used to model the nonlinear behaviour of an electromechanical transducer linked to a micromechanical resonator on the block diagram level without losing the link to the physical domain. However, it also showed the limitations of TDF MoC concerning the simulation of block diagram models of physical systems. The fast execution of data flow models is achieved by avoiding any control of the numerical error and the absence of convergence iterations and dynamic time stepping features. It showed the need for a more adapted MoC, which offers a similar flexibility in describing the abstract behaviour of modules, but causes less numerical problems during simulation.

The new Bond Graph (BG) MoC implemented in the SCAX library using all these techniques fulfils these needs. It offers a solid base for the modelling and simulation of the conservative and non-conservative CT behavioural aspects of heterogeneous multiphysical systems. It integrates itself seamlessly with the Fraunhofer SystemC-AMS library and OSCI SystemC library by exclusively using the synchronisation layer of SystemC-AMS and thus avoiding any modifications to the base libraries. Bond graph and block diagram descriptions can be freely mixed with each other and use predefined and user-defined bond graph and block diagram primitives. These BG models can communicate directly through converter ports with the TDF and DE MoCs of SystemC-AMS. With the introduction of the bond graph formalism into the SystemC-AMS simulation framework, this work enables the *adapted*, *unified*, and *reusable* description of the energy conserving behavioural aspects of heterogeneous multiphysical systems like it has not been possible before. The BG MoC implements causality analysis at elaboration time to transform the bond graphs into an equivalent signal flow model that is merged for execution with the connected block diagram model. This enables their fast execution based on the static scheduling of the processing member functions describing the input/output behaviour of the block diagram and bond graph primitives. The BG MoC checks the convergence of the model after each iteration and performs, if necessary, fixed-point iterations to improve the precision of the solution for

5. Enhancing the SystemC AMS extensions for Multiphysical Systems Modelling

the current point in time before advancing the simulation with a variable time step. This procedural solution achieves an interesting simulation performance advantage over “classical” generalised network solvers that globally solve the DAE system extracted from the model. The improved convergence of bond graph and block diagram models employing the BG MoC over equivalent block diagram models using the TDF MoC has been demonstrated on the electromechanical transducer example, which allowed to increase the simulation time step by a factor of 200, which translated into a 50 times faster execution of the model at comparable precision. Together with the possible tight interaction with the other MoCs of the SystemC-AMS framework, this enables the implementation of high performance models of heterogeneous systems using adapted description formalisms for its individual analogue, RF, MEMS, digital, and software components. The causality and dependency analyses performed by the BG MoC at elaboration time give the designer a unique insight into the physical and computational structure of his models and enables other formal checks, which provide him with hints regarding modelling problems such as algebraic loops, multiple drivers, and ill-formed models. It helps him to cope with convergence problems observed for models containing algebraic loops by hinting him to the problematic region of his model. This allows a more focused analysis of the bond graph to find potential simplifications and/or review of the modelling assumptions to add neglected energy storing parasitics with the goal of obtaining a model with good numeric properties.

The SCAX library with its BG MoC forms a solid base for further developments to improve the modelling of multiphysical systems. It needs to be evaluated on more complex system examples than the ones used during its development and presented in this chapter. This might yield new requirements, which need to be addressed. Future work should target the improvement of the numerical algorithm employed to locally solve the algebraic constraints imposed by algebraic loops onto bond graph and block diagram models. This could be achieved by incorporating a suitable root-finding algorithm like the Newton-Raphson or secant methods into the implementation of the bond and directed signal classes. The root-finding algorithm could then be selectively activated on those variables, which are adjacent to the back edges of algebraic loops in the dependency graph of the processing member functions and their input/output variables registered by the BG modules. Another working direction is the development of a generic mathematical toolbox for the model author to further support him in the writing of BG modules by providing different numerical algorithms for integration, differentiation, filtering, etc., which interfaces are parameterisable to the quantity types used in the behavioural description and integrate well with the execution semantics of the BG MoC. Stronger support for the modelling of the interaction between the CT and DE domains is needed. This especially concerns the support for the switching of energy and signal flows inside bond graphs and block diagrams, respectively, due to digital events and the generation of digital events due to detected threshold crossings of CT signals. For the BG MoC this means the implementation of controlled junctions to describe hybrid bond graphs. This will also require work on the synchronisation layer of SystemC-AMS itself. This layer is missing standardisation from the side of the OSCI SystemC AMS extensions to offer an official API for the integration of third-party solvers and MoCs into the simulation framework. Another focus for future work can be put on the development and implementation of additional (semi)formal checks into the BG MoC to further audit the models at elaboration time regarding their dynamic and numerical properties to give the user insight into the modelled systems. Once the new concepts introduced with the SCAX library have been stabilised, their standardisation as part of the OSCI SystemC AMS extensions can be envisioned.

6. Conclusions and Outlook

This Ph.D. thesis described the author's contribution to the development of an efficient modelling and simulation methodology for heterogeneous Analogue and Mixed-Signal (AMS) Systems on Chip (SoCs) based on a common modelling and simulation platform for the system level specification, design, and verification. Today's challenges in the design process for such systems were addressed in three thematic phases.

In the first phase, the improvement of the reuse of existing models in the complex design process for heterogeneous systems has been addressed. To this end, a prototype of a web-based and thus CAD/EDA-tool-independent platform for the collection of models from different domains and levels of abstraction together with their associated structural and semantical meta information has been developed. This prototype is called *ModelLib*. Its web interface allows browsing for a model through the model class hierarchy. The meta information about models, test benches, design languages, design tools, and external documents that are stored in the meta information database can be displayed. New models can be added by committing them into the repository and adding/editing their meta information in the meta information database using the web interface. This work included the implementation of a hierarchical access control mechanism, which is able to protect the IP constituted by the models at different levels of detail. The use cases developed for this tool show how it can support the AMS SoC design process for heterogeneous systems by fostering the reuse and collaborative development of models for tasks like architecture exploration, system validation, and creation of more and more elaborated models of the system. The presentation of this early prototype yielded external interest and gave input in the form of new ideas and requirements to consider additional kinds of meta information and to enable a tighter tool integration. The ideas included the definition of a tool-independent model meta information exchange format, partially automated model import into the library using documentation extraction tools, and the definition of an extensible meta information data model, which would enable the storage of EDA-tool-specific information to support use cases such as hierarchical synthesis of AMS systems based on the models provided by *ModelLib*. These research perspectives and the identified unsatisfying separation of the application logic and presentation aspects of the PHP-based *ModelLib* web interface led to the decision to reimplement the *ModelLib* application following the 3-tier architecture concept based on the feature-rich Java EE platform. The usage of Java EE indeed allowed a better modularisation of the *ModelLib* application, but unfortunately proved also to be a burden on the prototype development. In particular, it turned out to be difficult to efficiently propagate the necessary changes of the evolving meta information data model, which addressed the new requirements, through the different architectural layers of the *ModelLib* prototype. The solution of these pure implementation-related problems were outside the scope of this work so that the *ModelLib* reimplementation remained unfinished.

However, the experiences from the *ModelLib* development delivered valuable insight into which aspects need to be especially addressed throughout the development of a model to make it reusable: mainly flexibility, documentation, and validation. This was the starting point for the development of an efficient modelling methodology for the top-down design and bottom-up verification of complex RF systems based on the systematic usage of behavioural models in the second phase. One outcome is the developed library of well documented, parameterisable, and pin-accurate VHDL-AMS models of typical analogue/digital/RF components of a transceiver. The models offer the designer two sets

6. Conclusions and Outlook

of parameters: one based on the performance specifications and one based on the device parameters back-annotated from the transistor level implementation. The abstraction level used for the description of the respective analogue/digital/RF component behaviour has been chosen to achieve a good trade-off between accuracy, fidelity, and simulation performance. The pin-accurate model interfaces facilitate the integration of transistor level models for the validation of the behavioural models or the verification of a component implementation in the system context. These properties make the models suitable for different design tasks such as architecture exploration or overall system validation. This has been demonstrated on a model of a binary FSK transmitter parameterised to meet very different target specifications. Its careful organisation allowed to achieve a full orthogonalisation of the *model structure*, *parametrisation*, and *abstraction selection* aspects avoiding code duplication as well as simplifying the addition of new design cases and model configurations for top-down architecture exploration. It can serve as a template for the implementation of other complex system level test benches. A systematic approach for the individual validation of transistor level component implementations in the system context has been presented, which achieves considerable gains in simulation performance compared to a full transistor level verification of the system. The development of the VHDL-AMS model library RF_TRX helped to establish best practices regarding the communication between model developers and RF designers as well as the organisation and documentation of the models. Their application ensures a maximum flexibility, reusability, validity, and maintainability of the models. This work shows that a complex RF system can be simulated rapidly and precisely by making the right abstraction choices. Future working directions include the consideration of different noise forms and the modelling of power consumption in the models as well as the development of new component models for the library to increase its coverage in the long term on the whole transceiver chain. However, for a complex heterogeneous SoC, the transceiver will constitute a just a single component of the overall system. Therefore, even the demonstrated behavioural modelling approach will show its limitations in terms of simulation performance and modelling capabilities of “classical” HDLs. New modelling formalisms supporting higher levels of abstractions are needed.

For this reason, the third and last phase was dedicated to further raise the abstraction level for the description of complex and heterogeneous AMS SoCs and thus enable their efficient simulation using different synchronised MoCs. This work is founded on the C++-based simulation framework SystemC with its AMS extensions. The author of this Ph.D. thesis actively contributed to the recent standardisation of these AMS extensions as member of the OSCI AMS Working Group and part of this research work. New modelling capabilities going beyond the standardised SystemC AMS extensions have been developed in form of the SystemC AMS extensions eXperiments (SCAX) library to describe the conservative and non-conservative continuous-time behaviour of heterogeneous multiphysical systems in a formal and consistent way at a high level of abstraction. Two suitable modelling formalisms were identified to support the description of these two kinds of behaviour: bond graphs and block diagrams, respectively. The generic nature of their modelling primitives over “classical” domain-specific modelling primitives make them more universally applicable. However, their usage requires a more precise specification of the model component interfaces and their calculations, to not lose the link to the physical domain, and a more careful assembly of the structural system model, to avoid interconnection errors especially at physical domain boundaries. To this end, all constants, variables, and parameters of the system model, which represent a physical quantity, can now declare their dimension and associated system of units as an intrinsic part of their data type. Assignments to them need to contain the correct measurement unit besides the numerical value. This allows a much more precise but still compact definition of the models’ interfaces and computations. Thus, the C++ compiler can check the correct assembly of the components and the coherency of the computations by means of *dimensional analysis*.

The implementation is based on the Boost.Units library, which employs template metaprogramming

techniques that only increase the compilation time but do not impose any simulation performance penalty. A dedicated filter for the measurement units data types has been implemented to simplify the compiler messages and thus facilitate the localisation of unit errors. To ensure the reusability of models despite precisely defined interfaces, their interfaces and behaviours need to be parametrisable in a well-defined manner. The enabling generic and functional programming techniques, to address these requirements, have been demonstrated with the developed library of generic block diagram component models for the Timed Data Flow (TDF) MoC of the SystemC AMS extensions. These techniques have been also the key to the implementation of a new *Bond Graph (BG) MoC*, which seamlessly integrates into Fraunhofer's proof-of-concept implementation of the SystemC AMS extensions and efficiently supports the parallel usage of the bond graph and block diagram formalisms in the same model description. An extensive library of predefined bond graph and block diagram primitives is provided, which can be augmented by user-defined primitives. The BG models can communicate directly through converter ports with the TDF and DE MoCs of the SystemC AMS simulation framework. With the introduction of the bond graph formalism into the SystemC AMS simulation framework, this work enables the *adapted, unified, and reusable* description of the energy conserving behavioural aspects of heterogeneous multiphysical systems like it has not been possible before. The BG MoC implements causality analysis at elaboration time to transform the bond graphs into an equivalent signal flow model that is merged for execution with the connected block diagram model. This enables their fast execution based on the static scheduling of the processing member functions describing the input/output behaviour of the block diagram and bond graph primitives. The BG MoC checks the convergence of the model after each iteration and performs, if necessary, fixed-point iterations to improve the precision of the solution for the current point in time before advancing the simulation with a variable time step. This procedural solution achieved, for the presented examples, good simulation performance advantages over "classical" generalised network solvers and the TDF MoC of the SystemC AMS extensions. Together with the possible tight interaction with the other MoCs of the SystemC AMS simulation framework, this enables the implementation of high performance models of heterogeneous systems using well adapted modelling formalisms for its individual analogue, RF, MEMS, digital, and software components. The causality and dependency analyses performed by the BG MoC at elaboration time give the designer a unique insight into the physical and computational structure of his models and enables other formal checks, which provide him with hints regarding modelling problems such as algebraic loops, multiple drivers, and ill-formed models. The SCAX library with its BG MoC forms a solid base for further developments to improve the modelling of multiphysical systems. Its novelty lies in the extension of the SystemC AMS simulation platform with new modelling capabilities that uniformly integrate all the aspects mentioned above and thus achieves the fusion of normally very opposing requirements: *genericity* to privilege reuse and *precision in the specification* to privilege verification. The SCAX library needs to be evaluated on more complex system examples than the ones used during its development and presented in this work. This might yield new requirements, which need to be addressed.

Future work should target the improvement of the numerical algorithm employed to locally solve the algebraic constraints imposed by algebraic loops onto bond graph and block diagram models. Another working direction is the development of a generic mathematical toolbox to facilitate the writing of user-defined BG primitives by providing different numerical algorithms for integration, differentiation, filtering, etc., which interfaces are parameterisable to the quantity types used in the behavioural description and integrate well with the execution semantics of the BG MoC. Stronger support for the modelling of the interaction between the CT and DE domains is needed to support the switching of energy and signal flows inside bond graphs and block diagrams, respectively, due to digital events and the generation of digital events due to detected threshold crossings of CT signals. This will

6. *Conclusions and Outlook*

also require work on the synchronisation layer of the SystemC AMS extensions itself, which currently lacks the standardisation of an official API for the integration of third-party solvers and MoCs into the simulation framework. Another focus for future work can be put on the development and implementation of additional (semi)formal checks into the BG MoC to further audit the models at elaboration time regarding their dynamic and numerical properties to give the user insight into the modelled system. Once the new concepts introduced with the SCAX library have been stabilised, their standardisation as part of the OSCI SystemC AMS extensions can be envisioned.

A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library

Table A.1.: Organisation of the SCAX library implementation into several namespaces. Each of the mentioned namespaces may contain a namespace detail for the implementation details.




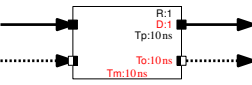






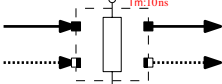








Namespace	#include header	Description
scax_core		Common functionality required by all parts of the SCAX library, i.e., for the moment the release information functions and tracing support for Boost.Units quantities in SystemC-AMS.
scax_util	scax_utility	<p>Utilities for the SystemC AMS extensions, which are not linked to a specific MoC but facilitate the development of generic modules (cf. to Section 5.6). This includes:</p> <ul style="list-style-type: none"> • Cast functions (<code>sc_time_cast<T>()</code> and <code>sca_time_cast<T>()</code>) to uniformly convert (back and forth) between the SystemC AMS extensions' time types (<code>sc_core::sc_time</code> and <code>sca_core::sca_time</code>) and a user specified data type <code>T</code> (e.g., double, <code>quantity<si::time></code>). • Data type traits (<code>scax_data_type_traits<T></code>) to obtain for the specified data type <code>T</code>, the data types of its value, unit, and system of units as well as its associated time data type. • Waveform functors (cf. to Table A.2) to generate common CT stimuli, e.g., exponential, pulse, and sinusoidal waves. • Functors for testing data sequences against a user-specified condition (e.g. threshold crossing) and react on its change. • Namespace <code>scax_util::math_functional</code>) with lazily evaluated function wrappers for the mathematical functions defined in <code><cmath></code> and <code><boost/units/cmath.hpp></code> for use in functors created with the Boost.Lambda library [84].
scax_tdf	scax_tdf	Generic block diagram modules for the TDF MoC (cf. to Table A.4 and to Section 5.6).
scax_bg	scax_bond_graph	New model of computation for the SystemC AMS extensions based on the bond graph formalism (cf. to Section 5.7) supporting also the block diagram formalism and synchronisation with the TDF and DE MoCs. A set of generic bond graph and block diagram primitives (cf. to Tables A.5 and A.6) is provided. The user can also define new bond graph and block diagram primitives.

A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library

Table A.2.: Generic waveform functors provided by the `scax_utility` library in the namespace `scax_util`. The return type of each functor has to be specified with the template parameter `T` upon its instantiation.

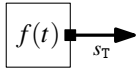

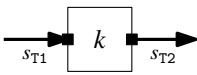
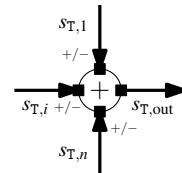
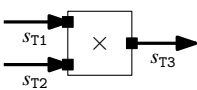
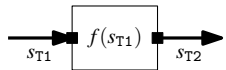
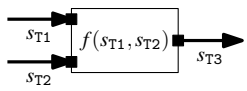
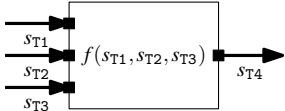
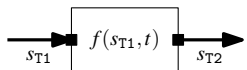
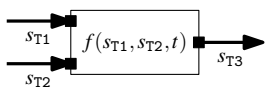
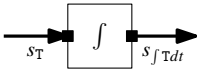
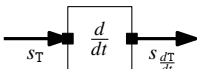
Functor name	Parameter	Description
<code>scax_am<T></code>	Amplitude Modulation (AM) waveform.	
	<code>amp / [T]</code>	Signal amplitude.
	<code>off / 1</code>	Offset of the modulation signal.
	<code>ws / rad/s or fs / Hz</code>	Angular velocity or frequency of the modulation signal.
	<code>wc / rad/s or fc / Hz</code>	Angular velocity or frequency of the carrier signal.
	<code>td / s</code>	Time delay before the signal starts.
<code>scax_constant<T></code>	Constant waveform.	
	<code>value / [T]</code>	Constant value.
<code>scax_exponential<T></code>	Exponential waveform.	
	<code>val_0 / [T]</code>	Initial value.
	<code>val_1 / [T]</code>	Target value.
	<code>td_1 / s</code>	First edge delay time.
	<code>tau_1 / s</code>	First edge time constant.
	<code>td_2 / s</code>	Second edge delay time.
	<code>tau_2 / s</code>	Second edge time constant.
<code>scax_pulse<T></code>	Periodic pulse waveform.	
	<code>v0 / [T]</code>	Initial value.
	<code>v1 / [T]</code>	Plateau value.
	<code>td / s</code>	Delay time before starting the pulse.
	<code>tr / s</code>	1st edge time (usually rising edge).
	<code>tf / s</code>	2nd edge time (usually falling edge).
	<code>pw / s</code>	Pulse width.
	<code>per / s</code>	Pulse period.
<code>scax_sffm<T></code>	Single frequency Frequency Modulation (FM) waveform.	
	<code>off / [T]</code>	Offset.
	<code>amp / [T]</code>	Amplitude.
	<code>wc / rad/s or fc / Hz</code>	Angular velocity or frequency of carrier.
	<code>mdi / rad</code>	Modulation index.
	<code>ws / rad/s or fs / Hz</code>	Angular velocity or frequency of signal.
<code>scax_sinusoidal<T></code>	Sinusoidal waveform.	
	<code>offset / [T]</code>	Offset value.
	<code>amplitude / [T]</code>	Amplitude.
	<code>omega / rad/s or frequency / Hz</code>	Angular velocity or frequency.
	<code>delay / s</code>	Delay time to start the sinusoidal variation.
	<code>theta / 1/s</code>	Damping factor.
	<code>phase / rad or phase / °</code>	Phase delay.

Table A.3.: Symbols used to represent the DE, TDF, LSF, ELN, and BG MoC modelling elements in schematics. Specified TDF attributes are denoted in **red** and propagated ones in black.

MoC	Symbol	Description	C++ classes
Discrete Event (DE)		Behavioural Discrete Event (DE) module (with ports and signals).	<code>sc_core::sc_module</code>
		DE port.	<code>sc_core::sc_in<T></code> <code>sc_core::sc_inout<T></code> <code>sc_core::sc_out<T></code>
		DE signal.	<code>sc_core::sc_signal<T></code>
Timed Data Flow (TDF)		Timed Data Flow (TDF) module (with allowed ports and signals) and with attribute time step T_m .	<code>sca_tdf::sca_module</code>
		TDF port with attributes: rate R (default: 1), delay D (default: 0), time step T_p , time offset T_o (default: 0s).	<code>sca_tdf::sca_in<T></code> <code>sca_tdf::sca_out<T></code>
		TDF converter port with attributes: rate R (default: 1), delay D (default: 0), time step T_p , time offset T_o (default: 0s).	<code>sca_tdf::sca_de::sca_in<T></code> <code>sca_tdf::sc_in<T></code> <code>sca_tdf::sca_de::sca_out<T></code> <code>sca_tdf::sc_out<T></code>
		TDF signal.	<code>sca_tdf::sca_signal<T></code>
Linear Signal Flow (LSF)		Linear Signal Flow (LSF) module (with allowed ports and signals) and with attribute time step T_m .	<code>sca_lsf::sca_module</code> (only available as predefined primitives)
		LSF port.	<code>sca_lsf::sca_in</code> <code>sca_lsf::sca_out</code>
		LSF signal.	<code>sca_lsf::sca_signal</code>
Electrical Linear Network (ELN)		Electrical Linear Network (ELN) module (with allowed ports and signals) and with attribute time step T_m .	<code>sca_eln::sca_module</code> (only available as predefined primitives)
		ELN terminal.	<code>sca_eln::sca_terminal</code>
		ELN node.	<code>sca_eln::sca_node</code>
		ELN reference node (ground).	<code>sca_eln::sca_node_ref</code>
Bond Graph (BG) with block diagram support		Bond Graph (BG) module (with allowed ports and signals) and with attribute time step T_m for synchronisation with the TDF and DE MoC via TDF (converter) ports. Only block diagram primitives show the block as part of their symbol, bond graph primitives (e.g., R , C , I) omit it and just have the ports next to their symbol.	<code>scax_bg::scax_module</code>
		BG bond port (a.k.a. power port) to be bound to the head or tail of a bond.	<code>scax_bg::scax_port<Domain></code>
		BG (power) bond (bidirectional channel for effort e and flow f variables), which two sides <i>head</i> and <i>tail</i> have to be bound to exactly one bond port.	<code>scax_bg::scax_bond<Domain></code>
		BG directed signal port for the block diagram formalism also supported by the BG MoC.	<code>scax_bg::scax_in<T></code> <code>scax_bg::scax_out<T></code>
		BG directed signal for the block diagram formalism also supported by the BG MoC.	<code>scax_bg::scax_signal<T></code>

A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library

Table A.4.: Generic block diagram modules for the TDF model of computation provided by the `scax_tdf` library in the namespace with the same name.

Name	Symbol	Description
<code>scax_source<T,TimeType></code>		TDF samples source module using a waveform function: $f : \text{TimeType} \rightarrow T$
<code>scax_sink<T></code>		TDF samples sink module.
<code>scax_scale<T1,T2></code>		Scale module.
<code>scax_sum<T></code>		Summing module with variable input number.
<code>scax_mul<T1,T2></code>		Multiplier module with two inputs.
<code>scax_func1<T1,T2></code>		Time-independent function module with one input: $f : T1 \rightarrow T2$
<code>scax_func2<T1,T2,T3></code>		Time-independent function module with two inputs: $f : T1 \times T2 \rightarrow T3$
<code>scax_func3<T1,T2,T3,T4></code>		Time-independent function module with three inputs: $f : T1 \times T2 \times T3 \rightarrow T4$
<code>scax_func1t<T1,T2,TimeType></code>		Time-dependent function module with one input: $f : T1 \times \text{TimeType} \rightarrow T2$
<code>scax_func2t<T1,T2,T3,TimeType></code>		Time-dependent function module with two inputs: $f : T1 \times T2 \times \text{TimeType} \rightarrow T3$
<code>scax_integ_trapez<T></code>		Trapezoidal integrator module.
<code>scax_dot_secant<T></code>		Differentiator module using asymmetric evaluation of Newton's difference quotient.

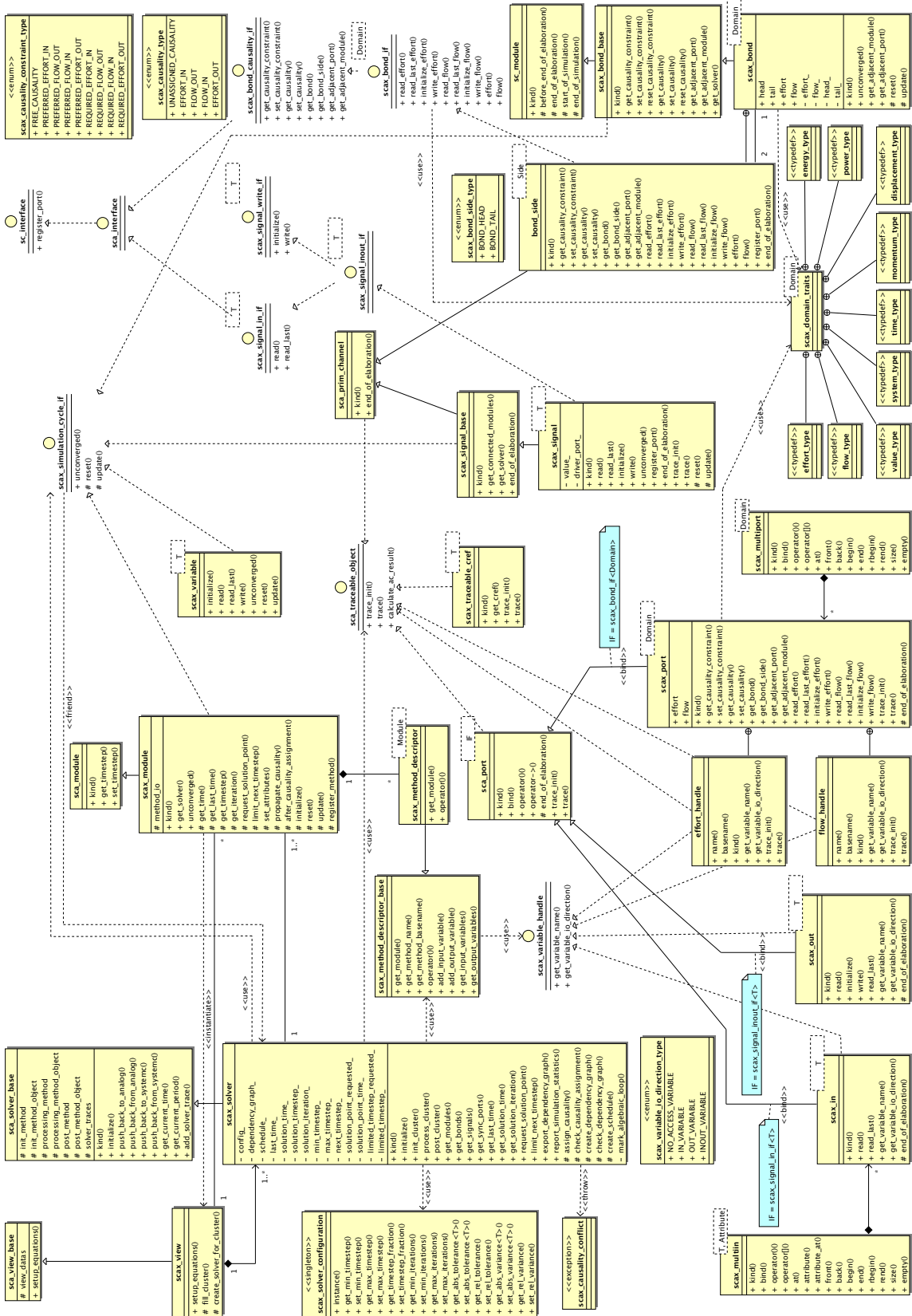


Figure A.1.: Diagram of the core classes of SCAX's BG MoC for Fraunhofer's SystemC-AMS.

A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library

Table A.5.: Generic block diagram modules for the BG model of computation provided by the scax_bond_graph library in the namespace scax_bg.

Name	Symbol	Description
scax_source<T,TimeType>		Directed signal source module using a waveform function: $f : \text{TimeType} \rightarrow T$
scax_de_source<T,TimeType>		DE samples to directed signal conversion module.
scax_tdf_source<T,TimeType>		TDF samples to directed signal conversion module.
scax_sink<T>		Directed signal sink module.
scax_de_sink<T>		Directed signal to DE samples conversion module.
scax_tdf_sink<T>		Directed signal to TDF samples conversion module.
scax_scale<T1,T2>		Scale module
scax_sum<T>		Summing module with variable input number.
scax_mul<T1,T2>		Multiplier module with two inputs.
scax_func1<T1,T2>		Time-independent function module with one input: $f : T1 \rightarrow T2$
scax_func2<T1,T2,T3>		Time-independent function module with two inputs: $f : T1 \times T2 \rightarrow T3$
scax_func3<T1,T2,T3,T4>		Time-independent function module with three inputs: $f : T1 \times T2 \times T3 \rightarrow T4$
scax_func1t<T1,T2,TimeType>		Time-dependent function module with one input: $f : T1 \times \text{TimeType} \rightarrow T2$
scax_func2t<T1,T2,T3,TimeType>		Time-dependent function module with two inputs: $f : T1 \times T2 \times \text{TimeType} \rightarrow T3$
scax_integ_trapez<T>		Trapezoidal integrator module.
scax_dot_secant<T>		Differentiator module using asymmetric evaluation of Newton's difference quotient.

Table A.6.: Generic bond graph modules for the BG MoC provided by the `scax_bond_graph` library in the namespace `scax_bg`. Note, the bond direction is not imposed by the bond graph modules. However, their parameter definition assumes the indicated bond directions.

Name	Symbol	Causality constraints	Parameter options
Time-dependent effort source:			
<code>scax_Se<D></code>		• e_{out} required	• Function $e_D(t)$
Modulated effort sources:			
<code>scax_MSe<D></code>		• e_{out} required	
<code>scax_de_MSe<D></code>		• e_{out} required	
<code>scax_tdf_MSe<D></code>		• e_{out} required	
Time-dependent flow source:			
<code>scax_Sf<D></code>		• f_{out} required	• Function $f_D(t)$
Modulated effort sources:			
<code>scax_MSf<D></code>		• f_{out} required	
<code>scax_de_MSf<D></code>		• f_{out} required	
<code>scax_tdf_MSf<D></code>		• f_{out} required	
Effort detectors:			
<code>scax_De<D></code>		• e_{in} required	
<code>scax_de_De<D></code>		• e_{in} required	
<code>scax_tdf_De<D></code>		• e_{in} required	
Flow detectors:			
<code>scax_Df<D></code>		• f_{in} required	
<code>scax_de_Df<D></code>		• f_{in} required	
<code>scax_tdf_Df<D></code>		• f_{in} required	
(Generalised) resistor:			
<code>scax_R<D></code>		• Free causality	<ul style="list-style-type: none"> • Constant $R_D = \frac{e_D}{f_D}$ • Functions $e_D = \Phi_R(f_D)$ and $f_D = \Phi_R^{-1}(e_D)$
Continued on next page...			

A. Short Reference for the SystemC AMS extensions eXperiments (SCAX) Library

Name	Symbol	Causality constraints	Parameter options
(Generalised) capacitor:			
scax_C<D>		<ul style="list-style-type: none"> f_{in} preferred 	<ul style="list-style-type: none"> Constant $C_D = \frac{q_D}{e_D}$ Functions $q_D = \Phi_C(e_D)$ and $e_D = \Phi_C^{-1}(q_D)$
(Generalised) inertia:			
scax_I<D>		<ul style="list-style-type: none"> e_{in} preferred 	<ul style="list-style-type: none"> Constant $I_D = \frac{p_D}{f_D}$ Functions $p_D = \Phi_I(f_D)$ and $f_D = \Phi_I^{-1}(p_D)$
(Generalised) transformer:			
scax_TF<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow e_{out,D2}$ $f_{in,D1} \longleftrightarrow f_{out,D2}$ 	<ul style="list-style-type: none"> Constant $m_{D1,D2} = \frac{e_{D1}}{e_{D2}} = \frac{f_{D2}}{f_{D1}}$
Modulated (generalised) transformers:			
scax_MTF<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow e_{out,D2}$ $f_{in,D1} \longleftrightarrow f_{out,D2}$ 	
scax_de_MTF<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow e_{out,D2}$ $f_{in,D1} \longleftrightarrow f_{out,D2}$ 	
scax_tdf_MTF<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow e_{out,D2}$ $f_{in,D1} \longleftrightarrow f_{out,D2}$ 	
(Generalised) gyrator:			
scax_GY<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow f_{out,D2}$ $f_{in,D1} \longleftrightarrow e_{out,D2}$ 	<ul style="list-style-type: none"> Constant $r_{D1,D2} = \frac{e_{D1}}{f_{D2}} = \frac{e_{D2}}{f_{D1}}$
Modulated (generalised) gyrators:			
scax_MGY<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow f_{out,D2}$ $f_{in,D1} \longleftrightarrow e_{out,D2}$ 	
scax_de_MGY<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow f_{out,D2}$ $f_{in,D1} \longleftrightarrow e_{out,D2}$ 	
scax_tdf_MGY<D1,D2>		<ul style="list-style-type: none"> $e_{in,D1} \longleftrightarrow f_{out,D2}$ $f_{in,D1} \longleftrightarrow e_{out,D2}$ 	

Continued on next page...

Name	Symbol	Causality constraints	Parameter options
Flow junction, 0-junction, common effort junction:			
scax_J0 <D>		<ul style="list-style-type: none"> • $e_{in,D,i} \longleftrightarrow e_{out,D,j \neq i}$ 	
Effort junction, 1-junction, common flow junction:			
scax_J1 <D>		<ul style="list-style-type: none"> • $f_{in,D,i} \longleftrightarrow f_{out,D,j \neq i}$ 	

Bibliography

- [1] David Abrahams and Aleksey Gurtovoy. *C++ Template Metaprogramming*. C++ In-Depth Series. Addison-Wesley Professional, Dec. 10, 2004. ISBN: 0-321-22725-5 (cit. on p. 92).
- [2] Accellera. *Verilog-AMS Language Reference Manual Version 2.3.1: Analog & Mixed-Signal Extensions to Verilog HDL*. Accellera. 1370 Trancas Street, #163, Napa, CA 94558, USA, June 1, 2009. URL: <http://www.verilog.org/verilog-ams/htmlpages/public-docs/lrm/2.3.1/VAMS-LRM-2-3-1.pdf> (visited on 10/10/2010) (cit. on pp. 3, 8, 10, 70, 73).
- [3] Accellera Verification Intellectual Property Technical Subcommittee (UVM). *Universal Verification Methodology (UVM) 1.0 Class Reference*. Accellera. 1370 Trancas Street #163, Napa, CA 94558, USA, Feb. 2011. URL: http://www.accellera.org/activities/vip/UVM_Class_Reference_Manual_1.0.pdf (visited on 03/22/2011) (cit. on p. 10).
- [4] Accellera Verification Intellectual Property Technical Subcommittee (UVM). *Universal Verification Methodology (UVM) 1.0 User's Guide*. Accellera. 1370 Trancas Street #163, Napa, CA 94558, USA, Feb. 23, 2011. URL: <http://www.accellera.org/activities/vip/uvm-1.0p1.tar.gz> (visited on 03/22/2011) (cit. on p. 10).
- [5] Sumit Adhikari and Christoph Grimm. “Modeling Switched Capacitor Sigma Delta Modulator Nonidealities in SystemC-AMS”. In: *Proceedings of the 13th International Forum on specification & Design Languages (FDL) 2010*. (Southampton, UK, Sept. 14–16, 2010). Ed. by Adam Morawiec and Jinnie Hinderscheit. ECSI. 2, Avenue de Vignate, Parc Equation, 38610 Gières, France (cit. on p. 15).
- [6] Adobe Systems Inc. et al. *Boost.TypeTraits*. 2000–2006. URL: http://www.boost.org/doc/libs/1_45_0/libs/type_traits/doc/html/index.html (visited on 01/14/2011) (cit. on p. 92).
- [7] The Apache Software Foundation. *ApacheTM Subversion[®]: Enterprise-class centralized version control for the masses*. The Apache Software Foundation. 2010. URL: <http://subversion.apache.org/> (visited on 12/12/2010) (cit. on pp. 25, 28).
- [8] The Apache Software Foundation. *The Apache HTTP Server Project*. The Apache Software Foundation. 1999–2009. URL: <http://httpd.apache.org/> (visited on 10/10/2010) (cit. on p. 26).
- [9] Rui Esteves Araújo, Américo Vicente Leite, and Diamantino Freitas. “Modelling and simulation of power electronic systems using bond-graph formalism in SIMULINK environment”. In: *Proceedings of the 10th IEEE Mediterranean Conference on Control and Automation (MED) 2002*. (Lisboa, Portugal, July 8–12, 2002). IEEE. URL: https://bibliotecadigital.ipb.pt/bitstream/10198/2225/1/avtl_MED02.pdf (visited on 01/05/2011) (cit. on p. 16).
- [10] Thomas Arndt et al. “Using SystemC-AMS for Heterogeneous Systems Modelling at TIER-1 Level”. In: *Proceedings of the 13th International Forum on specification & Design Languages (FDL) 2010*. (Southampton, UK, Sept. 14–16, 2010). Ed. by Adam Morawiec and Jinnie Hinderscheit. ECSI. 2, Avenue de Vignate, Parc Equation, 38610 Gières, France (cit. on p. 15).

- [11] Modelica Association. *Modelica: Modeling of Complex Physical Systems*. Modelica Association. 1997–2010. URL: <http://www.modelica.org/> (visited on 10/12/2010) (cit. on pp. 17, 19).
- [12] Ken Bakalar et al. *Meeting Minutes of September 19–22, 1995*. Minutes. Brighton, UK: IEEE 1076.1 Language Design Committee, Sept. 19–22, 1995 (cit. on p. 70).
- [13] F. Balarin et al. “Metropolis: an integrated electronic system design environment”. In: *IEEE Computer* 36 (4 Apr. 2003), pp. 45–52. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1193228 (cit. on p. 8).
- [14] Martin Barnasconi et al. *SystemC AMS extensions User’s Guide*. Open SystemC Initiative (OSCI). Mar. 8, 2010. URL: http://www.systemc.org/members/download_files/check_file?agreement=ams_ext_10std (visited on 10/12/2010) (cit. on pp. 14, 72, 73, 84).
- [15] Ntongo Christian Bazungula. “Development of a high-level VHDL-AMS simulation model of an RF transmitter based on a direct modulation architecture”. MA thesis. EPFL/STI/IEL/LSM, Station 11, CH-1015 Lausanne, Switzerland: École Polytechnique Fédérale de Lausanne (EPFL), Jan. 16, 2009 (cit. on pp. 53, 58).
- [16] Christopher D. Beers et al. “Building Efficient Simulations from Hybrid Bond Graph Models”. In: *Proceedings of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems*. (Alghero, Italy, June 7–9, 2006), pp. 71–76. URL: <http://www.diee.unica.it/~adhs06/CD/PAPERS/WB1.3.pdf> (visited on 01/05/2011) (cit. on pp. 16, 88, 110).
- [17] K. Besbes. “Modelling semiconductor devices using bond graph techniques”. In: *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE) 1997*. (Guimaraes, July 7–11, 1997). Vol. 2, pp. 201–205. DOI: 10.1109/ISIE.1997.648934. URL: <http://ieeexplore.ieee.org/iel4/5230/14143/00648934.pdf> (visited on 01/05/2011) (cit. on p. 16).
- [18] G. Biagetti et al. “Extending SystemC to analog modeling and simulation”. In: *Languages for System Specification. Selected Contributions on UML, SystemC, System Verilog, Mixed-Signal Systems, and Property Specifications from FDL’03*. Ed. by Christoph Grimm. ChDL. Dordrecht, The Netherlands: Springer, 2004, pp. 229–242. ISBN: 978-1-4020-7990-0 (cit. on p. 11).
- [19] P. Bjur  us and A. Jantsch. “Modeling of Mixed Control and Dataflow Systems in MASCOT”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9 (5 Oct. 2001), pp. 690–703. ISSN: 1063-8210. DOI: 10.1109/92.953502 (cit. on p. 8).
- [20] David C. Black et al. *SystemC: From the Ground Up*. 2nd ed. New York, Dordrecht, Heidelberg, London: Springer, 2010. ISBN: 978-0-387-69957-8 (cit. on pp. 8, 9, 71).
- [21] Thomas B  hm. “Development of a Web-Based Application for Collecting Models and Supporting the Design of AMS Systems”. Diplomarbeit. Fakult  t f  r Informatik (FIN), Postfach 4120, D-39016 Magdeburg: Otto-von-Guericke-Universit  t Magdeburg, Feb. 1, 2007 (cit. on pp. 31, 33).
- [22] T. E. Bonnerud, B. Hernes, and T. Ytterdal. “A mixed-signal, functional level simulation framework based on SystemC for system-on-a-chip applications”. In: *Proceedings of the IEEE Conference on Custom Integrated Circuits (CICC) 2001*. (San Diego, CA, May 6–9, 2001). IEEE, pp. 541–544. ISBN: 0-7803-6591-7. DOI: 10.1109/CICC.2001.929838 (cit. on p. 11).
- [23] Matthew Bowen. *Handel-C Language Reference Manual Version 2.1*. Embedded Solutions Limited. URL: <http://www.pa.msu.edu/hep/d0/l2/Handel-C/Handel%20C.PDF> (visited on 01/08/2011) (cit. on p. 8).

- [24] Peter Breedveld. “Bond Graphs”. In: *Proceedings of the Geoplex Summer School 2003*. (Bertinoro, Italy, July 2003), pp. 1–27. URL: http://www-lar.deis.unibo.it/euron-geoplex-sumsch/files/lectures_1/Breedveld/Breedveld_03_BGConcepts.pdf (visited on 01/05/2011) (cit. on p. 16).
- [25] David Broman, Peter Aronsson, and Peter Fritzson. “Design Considerations for Dimensional Inference and Unit Consistency Checking in Modelica”. In: *Proceedings of the 6th International Modelica Conference*. (Bielefeld, Germany, Mar. 3–4, 2008). The Modelica Association, pp. 3–12. URL: <https://www.modelica.org/events/modelica2008/Proceedings/sessions/session1a1.pdf> (visited on 01/05/2011) (cit. on p. 71).
- [26] I. N. Bronstein et al. *Taschenbuch der Mathematik*. German. 5th ed. Thun und Frankfurt am Main: Verlag Harri Deutsch, 2000. ISBN: 3-8171-2015-X (cit. on p. 110).
- [27] Cadence Design Systems and Mentor Graphics, Inc. *Open Verification Methodology User’s Guide – Product Version 2.0*. OVM World. USA, Sept. 2008. URL: <http://www.ovmworld.org/> (visited on 03/22/2011) (cit. on p. 10).
- [28] Cadence Design Systems and Mentor Graphics, Inc. *OVM Class Reference – Version 2.0*. OVM World. USA, Sept. 2008. URL: <http://www.ovmworld.org/> (visited on 03/22/2011) (cit. on p. 10).
- [29] Cadsim Engineering. *CAMP-G: The Universal BondgraphTM Preprocessor for Modeling and Simulation of Mechatronics Systems*. Cadsim Engineering. 1993–2010. URL: <http://www.bondgraph.com/> (visited on 01/10/2011) (cit. on p. 16).
- [30] Ken Caluwaerts, Dimitri Galayko, and Philippe Basset. “SystemC-AMS Heterogeneous Modeling of a Capacitive Harvester of Vibration Energy”. In: *Proceedings of the IEEE International Behavioral Modeling and Simulation Workshop (BMAS) 2008*. (San José, CA, USA, Sept. 25–26, 2008). IEEE. URL: http://www.bmas-conf.org/2008/5-4_Paper.pdf (visited on 01/06/2011) (cit. on p. 13).
- [31] François E. Cellier. “World Wide Web – The Global Library: A Compendium of Knowledge About Bond Graph Research”. In: *Proceedings of the 3rd International Conference on Bond Graph Modeling and Simulation (ICBGM) 1997*. (Sheraton-Crescent Hotel, Phoenix, Arizona, Jan. 12–15, 1997). Society for Computer Simulation International (SCS). URL: <http://www.ece.arizona.edu/~cellier/bg.html> (visited on 01/05/2011) (cit. on p. 16).
- [32] François E. Cellier. *BondLib: Bond Graph Library*. Modeling and Simulation Research Group, Department of Computer Science, ETH Zurich. 1987–2007. URL: http://people.inf.ethz.ch/fcellier/Soft/Soft_index_engl.html (visited on 01/10/2011) (cit. on p. 17).
- [33] François E. Cellier and Robert T. McBride. “Object-Oriented Modeling of Complex Physical Systems Using the Dymola Bond-Graph Library”. In: *Proceedings of the 6th SCS International Conference on Bond Graph Modeling and Simulation (ICBGM) 2003*. (Orlando, Florida, Jan. 19–23, 2003). SCS, pp. 157–162. URL: http://www.inf.ethz.ch/personal/fcellier/Pubs/BG/icbgm_03_bglib.pdf (visited on 01/05/2011) (cit. on p. 17).
- [34] François E. Cellier and Àngela Nebot. “The Modelica Bond Graph Library”. In: *Proceedings of the 4th International Modelica Conference*. (Hamburg, Germany, Mar. 7–8, 2005). Vol. 1. The Modelica Association, pp. 57–65. URL: http://www.inf.ethz.ch/personal/fcellier/Pubs/BG/modelica_05.pdf (visited on 01/05/2011) (cit. on p. 17).

- [35] Srikanth Chandrasekaran. *Verilog-AMS Integration with P1800 SV Standard*. Requirements presentation. Accellera Verilog Analog Mixed-Signal Group, Feb. 10, 2010. URL: http://www.verilog.org/verilog-ams/htmlpages/public-docs/SystemVerilogMerge/P1800_VerilogAMS_Requirements_Feb10.ppt (visited on 01/11/2011) (cit. on p. 10).
- [36] Jesse E. Chen. *Modeling RF Systems*. Tech. rep. The Designer's Guide Community, Mar. 6, 2005. URL: <http://www.designers-guide.org/Modeling/modeling-rf-systems.pdf> (visited on 10/11/2010) (cit. on p. 47).
- [37] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. 2nd ed. Sebastopol, USA: O'Reilly Media, Inc., Sept. 23, 2008. URL: <http://svnbook.red-bean.com/> (visited on 10/11/2010) (cit. on pp. 25, 28).
- [38] Controllab Products B.V. *20-sim: The Software for Modeling Dynamic Systems*. Controllab Products B.V. 1995–2010. URL: <http://www.20sim.com/> (visited on 01/10/2011) (cit. on p. 16).
- [39] Markus Damm et al. “Connecting SystemC-AMS models with OSCI TLM 2.0 models using temporal decoupling”. In: *Proceedings of the 11th International Forum on specification & Design Languages (FDL) 2008*. (Stuttgart, Germany, Sept. 23, 2008–Sept. 25, 2009). ECSI. ISBN: 978-1-4244-2266-1. DOI: 10.1109/FDL.2008.4641416 (cit. on p. 13).
- [40] Dassault Systèmes. *Dymola: Multi-Engineering Modeling and Simulation*. Dassault Systèmes. 2002–2011. URL: <http://www.dymola.com/> (visited on 01/08/2011) (cit. on pp. 17, 71).
- [41] Beman Dawes, David Abrahams, Rene Rivera, et al. *Boost C++ Libraries*. 1998–2010. URL: <http://www.boost.org/> (visited on 01/05/2011) (cit. on p. 97).
- [42] Beman Dawes, David Abrahams, Rene Rivera, et al. *Boost Library Documentation*. 1998–2010. URL: <http://www.boost.org/doc/libs> (visited on 01/05/2011) (cit. on pp. 100, 101).
- [43] Joel de Guzman and Hartmut Kaiser. *Spirit 2.4.1*. 2001–2010. URL: <http://www.boost.org/libs/spirit/doc/html/index.html> (visited on 01/13/2011) (cit. on p. 97).
- [44] B. De Smedt and G. Gielen. “Models for systematic design and verification of frequency synthesizers”. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 46 (10 Oct. 1999), pp. 1301–1308. ISSN: 1057-7130. DOI: 10.1109/82.799680 (cit. on p. 8).
- [45] Krister Edström, Jan-Erik Stromberg, and Jan Top. “Aspects on simulation of switched bond graphs”. In: *Proceedings of the 35th IEEE Conference on Decision and Control (CDC) 1996*. (Kobe, Japan, Dec. 11–13, 1996). Vol. 3, pp. 2642–2647. DOI: 10.1109/CDC.1996.573502. URL: <http://ieeexplore.ieee.org/iel3/4303/12402/00573502.pdf> (visited on 01/05/2011) (cit. on p. 16).
- [46] Karsten Einwich. “Application of SystemC/SystemC-AMS for the Specification of Complex Wired Telecommunication System”. In: *Proceedings of the 8th International Forum on specification and Design Languages (FDL) 2005*. (École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, Sept. 27–30, 2005). ECSI, pp. 49–60 (cit. on pp. 12, 14, 72, 84).
- [47] Karsten Einwich et al. “SystemC-AMS Extension Library for Modeling Conservative Nonlinear Dynamic Systems”. In: *Proceedings of the 9th Forum on Specification and Design Languages (FDL) 2006*. (Darmstadt University, Germany, Sept. 19–22, 2006). ECSI, pp. 113–120. ISBN: 978-3-00-019710-9 (cit. on pp. 14, 15).

- [48] Karsten Einwich et al. *Requirements Specification for SystemC Analog Mixed Signal (AMS) extensions*. Tech. rep. Version 2.1. Open SystemC Initiative (OSCI), Mar. 8, 2010. URL: http://www.systemc.org/members/download_files/check_file?agreement=ams_ext_10std (visited on 10/12/2010) (cit. on pp. 14, 72–74).
- [49] J. Eker et al. “Taming Heterogeneity – The Ptolemy Approach”. In: *Proceedings of the IEEE* 91 (1 Jan. 2003), pp. 127–144. ISSN: 0018-9219. doi: 10.1109/JPROC.2002.805829 (cit. on pp. 7, 73).
- [50] John Ellson et al. *Graphviz – Graph Visualization Software*. AT&T. 1996–2004. URL: <http://www.graphviz.org/> (visited on 01/19/2011) (cit. on p. 131).
- [51] Erhard Fehlbauer et al. *Dokumentation von VHDL-AMS-Modellen*. German. Version 1.0. Fraunhofer-Institut für Integrierte Schaltungen IIS, Außenstelle Entwurfsautomatisierung EAS. Zeunerstraße 38, DE-01069 Dresden, Germany, July 2004 (cit. on p. 19).
- [52] Jenny Montbrun-Di Filippo et al. “A survey of bond graphs : Theory, applications and programs”. In: *Journal of the Franklin Institute* 328.5-6 (1991), pp. 565–606. ISSN: 0016-0032. doi: 10.1016/0016-0032(91)90044-4 (cit. on p. 16).
- [53] Fraunhofer IIS/EAS. *Homepage of the SystemC-AMS Proof-of-Concept (PoC) implementation developed by Fraunhofer IIS/EAS*. Fraunhofer-Institut für Integrierte Schaltungen IIS, Institutsteil Entwurfsautomatisierung EAS. 2010. URL: <http://systemc-ams.eas.iis.fraunhofer.de/> (visited on 01/08/2011) (cit. on pp. 14, 74, 101, 109).
- [54] Ronny Frevert, Joachim Haase, and Roland Jancke. *Modeling and Simulation for RF System Design*. Berlin: Springer, Dec. 2005. ISBN: 978-0387275840 (cit. on pp. 10, 38).
- [55] Daniel Frey. *TextFilt*. Aug. 7–Sept. 5, 2002. URL: <http://textfilt.sourceforge.net/> (visited on 01/12/2011) (cit. on p. 95).
- [56] Thaddaeus Frogley. “An introduction to C++ Traits”. In: *Overload Journal* 43 (2001-06). URL: <http://accu.org/index.php/journals/442> (visited on 01/14/2011) (cit. on p. 92).
- [57] D. D. Gajski et al. *SpecC: Specification Language and Methodology*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2000. ISBN: 978-0-7923-7822-8 (cit. on p. 8).
- [58] Emden R. Gansner and Stephen C. North. “An open graph visualization system and its applications to software engineering”. In: *Software: Practice and Experience* 30 (11 2000-09): *Special Issue: Discrete algorithm engineering*, pp. 1203–1233. doi: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N. URL: <http://www.graphviz.org/Documentation/GN99.pdf> (visited on 01/19/2011) (cit. on p. 131).
- [59] Peter Gawthrop. *Model Transformation Tools (MTT)*. Department of Mechanical Engineering, Faculty of Engineering, Glasgow University. 2004. URL: <http://www.mech.gla.ac.uk/~peterg/software/MTT/> (visited on 01/10/2011) (cit. on p. 16).
- [60] Gert-Helge Geitner. *Add-on library BG V2.1 for graphical programming of Bondgraphs by means of Simulink*. ETI, TU Dresden. 2008. URL: http://eeiwzg.et.tu-dresden.de/ae2_files/ae_8_1e.htm (visited on 01/10/2011) (cit. on p. 17).
- [61] A. Gerstlauer et al. *System Design. A Practical Guide with SpecC*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2001. ISBN: 978-0-7923-7387-2 (cit. on p. 8).
- [62] Frank Ghenassia, ed. *Transaction-Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*. Springer, 2005. ISBN: 978-0-387-26232-1 (cit. on p. 9).

- [63] Georges G. E. Gielen. “CAD tools for embedded analogue circuits in mixed-signal integrated systems on chip”. In: *IEEE Proceedings – Computers and Digital Techniques* 152.3 (2005), pp. 317–332. ISSN: 1350-2387. DOI: 10.1049/ip-cdt:20045116 (cit. on pp. 5, 8, 38).
- [64] J.J. Granda and J. Reus. “New developments in bond graph modeling software tools: the computer aided modeling program CAMP-G and MATLAB”. In: *Computational Cybernetics and Simulation. Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics (ICSMC) 1997*. (Orlando, FL, USA, Oct. 12–15, 1997). Vol. 2, pp. 1542–1547. ISBN: 0-7803-4053-1. DOI: 10.1109/ICSMC.1997.638215 (cit. on p. 16).
- [65] Christoph Grimm et al. *An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions*. Tech. rep. OSCI, June 9, 2008. URL: http://www.systemc.org/downloads/standards/ams10/OSCI_Whitepaper_SystemC_AMS_extensions.pdf (visited on 10/11/2010) (cit. on p. 74).
- [66] T. Grötter et al. *System Design with SystemC*. Springer, 2002. ISBN: 978-1-4020-7072-3 (cit. on pp. 8, 9, 13, 71).
- [67] Joachim Haase. “Rules for Analog and Mixed-Signal VHDL-AMS Modeling”. In: *Proceedings of the Forum on Specification and Design Languages (FDL) 2003*. (Frankfurt/Main, Sept. 23–26, 2003). ECSI, 2003, pp. 98–107. URL: <http://www.eas.iis.fhg.de/publications/papers/2003/023/paper.pdf> (visited on 01/05/2011) (cit. on p. 15).
- [68] Joachim Haase. *Guidelines for the Development of a VHDL-AMS Model Library*. Draft Proposal. Version 2.0. Fraunhofer-Institut für Integrierte Schaltungen IIS, Außenstelle Entwurf-sautomatisierung EAS, Zeunerstraße 38, DE-01069 Dresden, Germany: VDA/FAT working group AK 30 on “Simulation of Mixed Systems with VHDL-AMS”, June 11, 2008. URL: http://fat-ak30.eas.iis.fraunhofer.de/pd/vhdl-ams_rules_2.0.pdf (visited on 10/11/2010) (cit. on p. 20).
- [69] A. Hajimiri and T. H. Lee. “A general theory of phase noise in electrical oscillators”. In: *IEEE Journal of Solid-State Circuits* 33.2 (Feb. 1998), pp. 179–194. ISSN: 0018-9200. DOI: 10.1109/4.658619 (cit. on p. 40).
- [70] P. A. Hartmann et al. “Modelling control systems in SystemC AMS – Benefits and limitations”. In: *Proceedings of the IEEE International SOC Conference (SOCC) 2009*. (Sept. 9–11, 2009), pp. 263–266. ISBN: 978-1-4244-4940-8. DOI: 10.1109/SOCCON.2009.5398043 (cit. on p. 15).
- [71] F. Herrera and E. Villar. “A framework for embedded system specification under different models of computation in SystemC”. In: *Proceedings of the 43rd ACM/IEEE Design Automation Conference (DAC) 2006*. (San Francisco, CA, USA, 2006). ACM/IEEE, pp. 911–914. ISBN: 1-59593-381-6. DOI: 10.1109/DAC.2006.229411 (cit. on pp. 9, 13).
- [72] F. Herrera, E. Villar, and C. Grimm. “A general approach to the interoperability of HetSC and SystemC-AMS”. In: *Proceedings of the 10th Forum on specification & Design Languages (FDL) 2007*. (Barcelona, Sept. 18–20, 2007). ECSI, pp. 18–24. ISBN: 978-2-9530504-0-0. URL: <http://www.ecsi-association.org/ecsi/fdl/fdl07/> (cit. on p. 13).
- [73] Ewald Hessel et al. “Development of VHDL-AMS-Libraries for Automotive Applications”. In: *Proceedings of the 8th International Forum on specification and Design Languages (FDL) 2005*. (École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, Sept. 27–30, 2005). ECSI, 2005, pp. 101–110 (cit. on pp. 19, 25).
- [74] HighTech Consultants. *Bondgraphs.com: Everything about Bondgraphs*. HighTech Consultants, 2007. URL: <http://bondgraph.org/> (visited on 01/10/2011) (cit. on p. 16).

- [75] HighTech Consultants. *SYMBOLS Sonata™*. HighTech Consultants. 2007. URL: <http://htcinfo.com/> (visited on 01/10/2011) (cit. on p. 16).
- [76] IEEE Computer Society. *1076.1-2007 IEEE Standard VHDL Analog and Mixed-Signal Extensions*. 1076.1-2007. IEEE. Nov. 15, 2007. ISBN: 0-7381-5627-2. DOI: 10.1109/IEEESTD.2007.4384309 (cit. on pp. 3, 8, 70, 73).
- [77] IEEE Computer Society. *1076-2008 IEEE Standard VHDL Language Reference Manual*. 1076-2008. IEEE. Jan. 26, 2009. ISBN: 978-0-7381-5801-3. DOI: 10.1109/IEEESTD.2009.4772740 (cit. on pp. 3, 8).
- [78] IEEE Computer Society. *IEC/IEEE Behavioural Languages – Part 4: Verilog Hardware Description Language (Adoption of IEEE Std 1364-2001)*. 1364-2001. IEEE. 2004. ISBN: 2-8318-7675-3. DOI: 10.1109/IEEESTD.2004.95753 (cit. on pp. 3, 8, 9).
- [79] IEEE Computer Society. *1666-2005 IEEE Standard SystemC Language Reference Manual*. 1666-2005. IEEE. Mar. 31, 2006. ISBN: 0-7381-4871-7. DOI: 10.1109/IEEESTD.2006.99475 (cit. on pp. 3, 8, 9, 71, 75, 99, 101, 111, 118, 120).
- [80] IEEE Computer Society. *1800-2005 IEEE Standard for System Verilog—Unified Hardware Design, Specification, and Verification Language*. 1800-2005. IEEE. 2005. ISBN: 0-7381-4810-5. DOI: 10.1109/IEEESTD.2005.97972 (cit. on p. 9).
- [81] IEEE Computer Society. *1800-2009 IEEE Standard for System Verilog—Unified Hardware Design, Specification, and Verification Language*. 1800-2009. IEEE. 2009. ISBN: 978-0-7381-6129-7. DOI: 10.1109/IEEESTD.2009.5354441 (cit. on p. 9).
- [82] Open SystemC Initiative (OSCI). *Homepage of the Open SystemC Initiative (OSCI)*. Open SystemC Initiative (OSCI). 2010. URL: <http://www.systemc.org/> (visited on 10/12/2010) (cit. on pp. 3, 9, 14, 101).
- [83] ITI GmbH. *SimulationX: Multi-domain System Simulation and Modeling*. ITI GmbH. 2002–2010. URL: <http://www.simulationx.com/> (visited on 01/08/2011) (cit. on p. 71).
- [84] Jaakko Järvi. *Boost.Lambda*. 1999–2004. URL: <http://www.boost.org/doc/html/lambda.html> (visited on 01/05/2011) (cit. on pp. 101, 107, 151).
- [85] Eric Jendrock et al. *The Java EE 5 Tutorial for Sun Java System Application Server Platform Edition 9*. Sun Microsystems. Sept. 2006. URL: <http://java.sun.com/javaee/5/docs/tutorial/doc/> (visited on 12/09/2010) (cit. on p. 33).
- [86] J. J. van Dixhoorn J. J. A. J. Beukeboom and J. W. Meerman. “Simulation of Mixed Bond Graphs and Block Diagrams on Personal Computer Using TUTSIM”. In: *Journal of the Franklin Institute* 319.1-2 (1985-01/1985-02), pp. 257–267. DOI: 10.1016/0016-0032(85)90079-1 (cit. on p. 16).
- [87] Paul M. Jones. *YaWiki: Yet Another Wiki for PHP*. 2005. URL: <http://www.yawiki.com/> (visited on 03/27/2006) (cit. on pp. 26, 28).
- [88] JTC1/SC22. *ISO/IEC 14977:1996(E). Information technology – Syntactic metalanguage – Extended BNF*. 14977. ISO/IEC. 1996. URL: [http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip) (visited on 01/13/2011) (cit. on p. 97).
- [89] JTC1/SC22/WG21 – The C++ Standards Committee. *ISO/IEC 14882:2003. Standard for Programming Language C++*. 14882. ISO/IEC. Oct. 16, 2003 (cit. on pp. 100, 101).

- [90] JTC1/SC22/WG21 – The C++ Standards Committee. *ISO/IEC DTR 19768: Draft Technical Report on C++ Library Extensions*. Tech. rep. N1836=05-0096. June 24, 2005. URL: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf> (visited on 01/15/2011) (cit. on pp. 103, 107).
- [91] Hessa Al-Junaid, Tom Kazmierski, and Leran Wang. “SystemC-A Modeling of an Automotive Seating Vibration Isolation System”. In: *Proceedings of the 9th Forum on Specification and Design Languages (FDL) 2006*. (Darmstadt University, Germany, Sept. 19–22, 2006). ECSI, pp. 107–112. ISBN: 978-3-00-019710-9 (cit. on pp. 12, 71).
- [92] H. Al-Junaid and T. Kazmierski. “Analogue and mixed-signal extension to SystemC”. In: *IEEE Proceedings on Circuits, Devices and Systems* 152 (6 2005), pp. 682–690 (cit. on p. 12).
- [93] D. C. Karnopp et al., eds. 308.3 (1979): *Bond Graph Techniques for Dynamic Systems in Engineering and Biology*. special issue (cit. on p. 16).
- [94] Dean C. Karnopp, Donald L. Margolis, and Ronald C. Rosenberg. *System Dynamics: Modeling and Simulation of Mechatronic Systems*. 4th ed. New York, USA: Wiley, Jan. 2006 (cit. on pp. 16, 84, 87, 88, 90, 91, 109, 125, 129, 140).
- [95] Dean C. Karnopp and Ronald C. Rosenberg. *Analysis and Simulation of Multiport Systems – The Bond Graph Approach to Physical Systems Dynamics*. Cambridge, Mass., USA: M.I.T. Press, 1968 (cit. on p. 16).
- [96] Dean C. Karnopp and Ronald C. Rosenberg. *System Dynamics: A Unified Approach*. 1st ed. New York, USA: John Wiley, 1974 (cit. on p. 16).
- [97] William O. Keese. *An Analysis and Performance Evaluation of a Passive Filter Design Technique for Charge Pump PLL’s*. Application Note 1001. 1111 West Bardin Road, Arlington, TX 76017, USA: National Semiconductor Corporation, May 1996. URL: <http://www.national.com/an/AN/AN-1001.pdf> (visited on 10/12/2010) (cit. on p. 42).
- [98] Andrew John Kennedy. “Types for Units-of-Measure: Theory and Practice”. In: *Proceedings of the 3rd Central European Functional Programming School (CEFP) 2009*. (Selye Janos University, Komarno, Slovakia, May 25–30, 2009) (cit. on p. 71).
- [99] Hans-Peter Kreuter et al. “System Level Modeling of Smart Power Switches using SystemC-AMS for Digital Protection Concept Verification”. In: *Proceedings of the 2009 IEEE International Behavioral Modeling and Simulation (BMAS) Workshop*. (Doubletree Hotel, San Jose, California, USA, Sept. 17–18, 2009). IEEE. 2009. URL: http://www.bmas-conf.org/2009/2-3_Paper.pdf (visited on 01/11/2011) (cit. on p. 15).
- [100] C. Lallement et al. “Compact Modeling of MOSFET in VHDL-AMS”. In: *Transistor Level Modeling for Analog/RF IC Design*. Ed. by W. Grabinski, B. Nauwelaers, and D. Schreurs. Springer, 2006. ISBN: 1-4020-4555-7 (cit. on p. 10).
- [101] Thomas H. Lee. *The Design of CMOS Radio-Frequency Integrated Circuits*. Cambridge, UK: Cambridge University Press, 1998. ISBN: 0-521-63061-4 (cit. on p. 42).
- [102] Ping Lu et al. “Mixed-Signal Test Development using Open Standard Modeling and Description Languages”. In: *Proceedings of the 2009 IEEE International Behavioral Modeling and Simulation (BMAS) Workshop*. (Doubletree Hotel, San Jose, California, USA, Sept. 17–18, 2009). IEEE. 2009. URL: http://www.bmas-conf.org/2009/4-1_Paper.pdf (visited on 01/11/2011) (cit. on p. 15).

- [103] John Maddock and Steve Cleary. “C++ Type Traits”. In: *Dr. Dobb's Journal* 25 (10 Oct. 1, 2000). URL: <http://www.drdobbs.com/184404270> (visited on 01/14/2011) (cit. on p. 92).
- [104] Torsten Mähne. *ModelLib Prototype*. Laboratoire de Systèmes Microélectroniques (LSM), École Polytechnique Fédérale de Lausanne (EPFL). July 13, 2006. URL: <https://lsmpc4.epfl.ch/modellib/> (visited on 10/11/2010) (cit. on p. 25).
- [105] Torsten Mähne and Alain Vachoux. “ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems”. In: *Proceedings of the 9th International Forum on Specification and Design Languages (FDL) 2006*. (Darmstadt University, Germany, Sept. 19–22, 2006). ECSI. 2006, pp. 91–97. ISBN: 978-3-00-019710-9. URL: <http://infoscience.epfl.ch/search.py?recid=88137> (visited on 10/11/2010) (cit. on p. 35).
- [106] Torsten Mähne and Alain Vachoux. “ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems”. In: *Advances in Design and Specification Languages for Embedded Systems. Selected Contributions from FDL'06*. Ed. by Sorin A. Huss. ChDL. Dordrecht, The Netherlands: Springer Verlag, July 2007. Chap. 4, pp. 53–72. ISBN: 978-1-4020-6147-9 (cit. on p. 25).
- [107] Torsten Mähne, Alain Vachoux, and Yusuf Leblebici. “Fostering the Reuse and Collaborative Development of Models in the AMS SoC Design Process”. In: *Proceedings of the 2007 Ph.D. Research in Microelectronics and Electronics (PRIME) Conference*. (Bordeaux, France, July 2–5, 2007). IEEE. 2007, pp. 285–288. ISBN: 1-4244-1000-2. DOI: 10.1109/RME.2007.4401868. URL: <http://prime07.ixl.fr/> (visited on 10/11/2010) (cit. on p. 35).
- [108] Torsten Mähne et al. “Creating Virtual Prototypes of Complex MEMS Transducers Using Reduced-Order Modelling Methods and VHDL-AMS”. In: *Applications of Specification and Design Languages for SoCs. Selected papers from FDL'05*. ChDL. Dordrecht, The Netherlands: Springer, 2006, pp. 135–153. ISBN: 1-4020-4997-8 (cit. on pp. 2, 10, 70).
- [109] E. Markert et al. “Untersuchung der Anwendbarkeit von SystemC-AMS bei der Beschreibung von MEMS”. German. In: *Tagungsband des 5. GMM/ITG/GI-Workshop „Multi-Nature-Systems: Optoelektronische, mechatronische und andere gemischte Systeme“*. (Dresden, Germany, Feb. 18, 2005). GMM, ITG, GI, pp. 13–18 (cit. on p. 12).
- [110] E. Markert et al. “SystemC-AMS Assisted Design of an Inertial Navigation System”. In: *IEEE Sensors Journal* 7.5 (May 2007), pp. 770–777. ISSN: 1530-437X. DOI: 10.1109/JSEN.2007.894130 (cit. on p. 12).
- [111] Deepak A. Mathaikutty et al. “UMoC++: A C++-Based Multi-MoC Modeling Environment”. In: *Applications of Specification and Design Languages for SoCs. Selected papers from FDL'05*. ChDL. Springer, 2006, pp. 115–130. ISBN: 1-4020-4997-8 (cit. on p. 9).
- [112] The MathWorks, Inc. *MATLAB - The Language Of Technical Computing*. The MathWorks, Inc. 1984–2011. URL: <http://www.mathworks.com/products/simulink/> (visited on 01/06/2011) (cit. on pp. 8, 17).
- [113] The MathWorks, Inc. *Simulink – Simulation and Model-Based Design*. The MathWorks, Inc. 1990–2011. URL: <http://www.mathworks.com/products/simulink/> (visited on 01/06/2011) (cit. on pp. 17, 73).
- [114] The MathWorks, Inc. *Simscape*. The MathWorks, Inc. 2008–2011. URL: <http://www.mathworks.com/products/simscape/> (visited on 01/06/2011) (cit. on p. 71).

- [115] Jan Mehner. *Entwurf in der Mikrosystemtechnik*. German. 1st ed. Vol. 9. Dresdner Beiträge zur Sensorik. Zugl.: Chemnitz, Techn. Univ., Habil., 1999. Dresden, München: Dresden University Press, 2000 (cit. on p. 7).
- [116] Mentor Graphics Corporation. *ADVance MS User's Manual: Release AMS 2008.2*. Mentor Graphics Corporation. 2008 (cit. on p. 65).
- [117] Mentor Graphics Corporation. *CommLib RF VHDL-AMS Library: Release AMS 2008.2*. Mentor Graphics Corporation. 2008 (cit. on pp. 19, 38).
- [118] Mentor Graphics Corporation. *Handel-C Synthesis Methodology*. Mentor Graphics Corporation. 2009–2011. URL: <http://www.mentor.com/products/fpga/handel-c/> (visited on 01/08/2011) (cit. on p. 8).
- [119] Wolfgang Menz and Jürgen Mohr. *Mikrosystemtechnik für Ingenieure*. German. 2nd ed. Postfach 101161, D-6940 Weinheim: VCH Verlagsgesellschaft, 1997 (cit. on p. 7).
- [120] N. Milet-Lewis et al. “A VHDL-AMS Library of RF Blocks Models”. In: *Proceedings of the Fifth IEEE International Workshop on Behavioral Modeling and Simulation (BMAS) 2001*. (Santa Rosa, CA, USA, Oct. 10–12, 2001). 2001, pp. 12–14. ISBN: 0-7803-7291-3. DOI: 10.1109/BMAS.2001.962490 (cit. on p. 38).
- [121] Modelica Association. *Modelica® – A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.2*. Modelica Association. Linköping, Sweden, Mar. 24, 2010. URL: <https://www.modelica.org/documents/ModelicaSpec32.pdf> (visited on 01/08/2011) (cit. on pp. 70, 71).
- [122] Eduard Moser and Wolfgang Nebel. “Case Study: System Model of Crane and Embedded Control”. In: *Proceedings of the Design, Automation and Test in Europe Conference (DATE) 1999*. (Munich, Germany, Mar. 9–12, 1999). Ed. by Dominique Borrione and Rolf Ernst. IEEE. 1999. URL: http://www.date-conference.com/proceedings/PAPERS/1999/DATE99/pdf/files/11b_1.pdf (visited on 01/11/2011) (cit. on p. 15).
- [123] Pieter J. Mosterman. “HyBrSim – A Modeling and Simulation Environment for Hybrid Bond Graphs”. In: *Journal of Systems and Control Engineering* 216.1 (Feb. 1, 2002), pp. 35–46. URL: <http://elib.dlr.de/11830/01/hybrsm-mosterman.pdf> (visited on 01/05/2011) (cit. on p. 16).
- [124] Wolfgang Müller, Wolfgang Rosenstiel, and Jürgen Ruf, eds. *SystemC. Methodologies and Applications*. Springer, 2003. ISBN: 978-1-4020-7479-0 (cit. on pp. 8, 9, 71).
- [125] Nathan C. Myers. “Traits: a new and useful template technique”. In: *C++ Report* (June 1995). URL: <http://www.cantrip.org/traits.html> (visited on 01/14/2011) (cit. on p. 92).
- [126] Lutz Nätke et al. “Hierarchical automatic behavioral model generation of nonlinear analog circuits based on nonlinear symbolic techniques”. In: *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE) 2004*. (CNIT, La Défense, Paris, France, Feb. 16–20, 2004). Vol. 1, pp. 442–447. DOI: 10.1109/DATE.2004.1268886 (cit. on p. 8).
- [127] James Oberg. “Why The Mars Probe Went Off Course”. In: *IEEE Spectrum Magazine* 36.12 (Dec. 1999) (cit. on p. 70).
- [128] Ian O'Connor et al. “UML/XML-Based Approach to Hierarchical AMS Synthesis”. In: *Applications of Specification and Design Languages for SoCs. Selected papers from FDL'05*. ChDL. Springer, 2006, pp. 205–225. ISBN: 1-4020-4997-8 (cit. on pp. 1, 35).

- [129] Simone Orcioni, Giorgio Biagetti, and Massimo Conti. “SystemC-WMS: Mixed-Signal Simulation Based on Wave Exchanges”. In: *Applications of Specification and Design Languages for SoCs. Selected Contributions from FDL’05*. Ed. by Alain Vachoux. ChDL. Dordrecht, The Netherlands: Springer, Oct. 2006. ISBN: 1-4020-4997-8 (cit. on pp. 11, 71).
- [130] OSCI AMS Working Group. *Draft Standard SystemC AMS Extensions Language Reference Manual*. Draft 1. Open SystemC Initiative (OSCI), Dec. 3, 2008. URL: http://www.systemc.org/members/download_files/check_file?agreement=AMS_draft1_120308 (visited on 10/12/2010) (cit. on pp. 14, 72).
- [131] OSCI AMS Working Group. *Standard SystemC AMS extensions Language Reference Manual*. Version 1.0. Open SystemC Initiative (OSCI). Mar. 8, 2010. URL: http://www.systemc.org/members/download_files/check_file?agreement=ams_ext_10std (visited on 10/12/2010) (cit. on pp. 3, 14, 72–74, 84, 88, 99, 101, 109, 111–113, 120, 125).
- [132] OSCI Analog/Mixed-signal Working Group (AMSWG). *Homepage of the Analog/Mixed-signal Working Group (AMSWG)*. Open SystemC Initiative (OSCI). 2006–2010. URL: http://www.systemc.org/apps/group_public/workgroup.php?wg_abbrev=amswg (visited on 01/05/2011) (cit. on pp. 14, 72).
- [133] OSCI TLM Working Group. *OSCI TLM-2.0 Language Reference Manual*. Version 2.0.1. Open SystemC Initiative (OSCI). July 2009. URL: http://www.systemc.org/members/download_files/check_file?agreement=tlm_2-0-1_090723 (visited on 01/11/2011) (cit. on p. 9).
- [134] M. Pastre, M. Kayal, and H. Blanchard. “A Hall sensor analog front end for current measurement with continuous gain calibration”. In: *Digest of Technical Papers of the IEEE International Solid-State Circuits Conference (ISSCC) 2005*. (San Francisco, CA, Feb. 6–10, 2005), pp. 242–259. DOI: 10.1109/ISSCC.2005.1493959 (cit. on p. 7).
- [135] H. D. Patel and S. K. Shukla. “Towards a heterogeneous simulation kernel for system-level models: a SystemC kernel for synchronous data flow models”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24 (8 Aug. 2005), pp. 1261–1271. ISSN: 0278-0070. DOI: 10.1109/TCAD.2005.850819 (cit. on pp. 9, 13).
- [136] Hiren D. Patel and Sandeep K. Shukla. *Ingredients for Successful System Level Design Methodology*. Springer, 2008. ISBN: 978-1-4020-8471-3 (cit. on p. 9).
- [137] Hiren Patel and Sandeep K. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modeling*. The Netherlands: Kluwer Academic Publishers, 2004. ISBN: 978-1402080876 (cit. on pp. 9, 71).
- [138] Henry M. Paynter. *Analysis and Design of Engineering Systems*. Cambridge, Mass., USA: M.I.T. Press, 1961 (cit. on p. 16).
- [139] Henry M. Paynter. “An Epistemic Prehistory of Bond Graphs”. In: *Bond Graphs for Engineers*. Ed. by P. C. Breedveld and G. Dauphin-Tanguy. Amsterdam: Elsevier, 1992, pp. 3–17 (cit. on p. 16).
- [140] F. Pêcheux, C. Lallement, and A. Vachoux. “VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24 (2 2005), pp. 204–225. DOI: 10.1109/TCAD.2004.841071 (cit. on p. 10).

- [141] François Pêcheux and Amr Habib. “Towards High-Level Executable Specifications of Heterogeneous Systems with SystemC-AMS: Application to a Manycore PCR-CE Lab on Chip for DNA Sequencing”. In: *Proceedings of the 13th International Forum on specification & Design Languages (FDL) 2010*. (Southampton, UK, Sept. 14–16, 2010). Ed. by Adam Morawiec and Jinnie Hinderscheit. ECSI. 2, Avenue de Vignate, Parc Equation, 38610 Gières, France (cit. on p. 15).
- [142] François Pêcheux et al. “Modeling and Simulation of Multi-Discipline Systems using Bond Graphs and VHDL-AMS”. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation (ICBGM) 2005*. (New Orleans, USA, Jan. 23–27, 2005). The Society for Modeling and Simulation International (SCS), pp. 149–155. URL: <http://infoscience.epfl.ch/getfile.py?mode=best&recid=54126> (visited on 01/05/2011) (cit. on p. 17).
- [143] The PHP Group. *PHP: Hypertext Preprocessor*. The PHP Group. 2001–2010. URL: <http://www.php.net/> (visited on 10/12/2010) (cit. on p. 26).
- [144] The PHP Group. *PEAR: The PHP Extension and Application Repository*. The PHP Group. 2001–2010. URL: <http://pear.php.net/> (visited on 10/12/2010) (cit. on p. 28).
- [145] J. L. Pino and K. Kalbasi. “Cosimulating synchronous DSP applications with analog RF circuits”. In: *Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems & Computers 1998*. (Pacific Grove, CA, Nov. 1–4, 1998). Vol. 2, pp. 1710–1714. DOI: 10.1109/ACSSC.1998.751617 (cit. on p. 8).
- [146] PostgreSQL Global Development Group. *PostgreSQL 8.1.22 Documentation*. The PostgreSQL Global Development Group. 1996–2010. URL: <http://www.postgresql.org/docs/8.1/interactive/index.html> (visited on 10/12/2010) (cit. on p. 31).
- [147] PostgreSQL Global Development Group. *PostgreSQL: The World’s Most Advanced Open Source Database*. PostgreSQL Global Development Group. 1996–2010. URL: <http://www.postgresql.org/> (visited on 10/12/2010) (cit. on pp. 25, 26).
- [148] Derek Robert Price. *CVS: Open Source Version Control*. Ximbiot. Dec. 3, 2006. URL: <http://www.nongnu.org/cvs/> (visited on 10/12/2010) (cit. on p. 28).
- [149] Vladimir Prus. *Boost.Program_options*. 2002–2004. URL: http://www.boost.org/doc/html/program_options.html (visited on 01/13/2011) (cit. on p. 97).
- [150] Monica Rafaila et al. “Design of Experiments for Reliable Operation of Electronics in Automotive Applications”. In: *Proceedings of the 13th International Forum on specification & Design Languages (FDL) 2010*. (Southampton, UK, Sept. 14–16, 2010). Ed. by Adam Morawiec and Jinnie Hinderscheit. ECSI. 2, Avenue de Vignate, Parc Equation, 38610 Gières, France (cit. on p. 15).
- [151] Jean Ravatin et al. “Full Transceiver Circuit Simulation Using VHDL-AMS”. In: *Microwave Engineering* (May 2002), pp. 29–33. URL: <http://www.bausch-gall.de/gsmtrans.pdf> (visited on 04/30/2009) (cit. on p. 38).
- [152] John Rogers, Calvin Plett, and Foster Dai. *Integrated Circuit Design for High-Speed Frequency Synthesis*. Artech House, Jan. 2006. ISBN: 978-1580539821 (cit. on pp. 40, 53).

- [153] Frank Rogin et al. “Automatische Generierung von Dokumentationen für VHDL-AMS-Modellbibliotheken”. German. In: *Simulations- und Testmethoden für Software in Fahrzeugsystemen (Proceedings der Jahrestagung der ASIM/GI-Fachgruppe 4.5.5 ,Simulation technischer Systeme‘)*. (Berlin, Mar. 1–2, 2005). Ed. by Mirko Conrad, Christoph Nytsch-Geusen, and Achim Wohnhaas. Technische Universität Berlin – Forschungsberichte der Fakultät IV 2005-1. URL: <http://swt.cs.tu-berlin.de/asim-sts-05/folien/rogin.pdf> (visited on 12/12/2010) (cit. on pp. 19, 25, 35).
- [154] Ronald C. Rosenberg. “Computer-Aided Teaching of Dynamic System Behavior”. PhD thesis. Boston, Mass., USA: M.I.T., 1965 (cit. on p. 16).
- [155] Ronald C. Rosenberg. *A User’s Guide to ENPORT-4*. New York, USA: Wiley, 1974 (cit. on p. 16).
- [156] SAE Electronic Design Automation Standards Committee. *Model Specification Process Standard*. J2546. 400 Commonwealth Drive, Warrendale, PA 15096-0001, USA, Feb. 1, 2002 (cit. on pp. 19, 22, 23).
- [157] I. Sander and A. Jantsch. “System modeling and transformational design refinement in ForSyDe [formal system design]”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23.1 (Jan. 2004), pp. 17–32. ISSN: 0278-0070. DOI: 10.1109/TCAD.2003.819898 (cit. on p. 8).
- [158] Matthias Christian Schabel and Steven Watanabe. *Boost.Units 1.1.0*. 2003–2010. URL: http://www.boost.org/doc/html/boost_units.html (visited on 01/05/2011) (cit. on pp. 92, 96, 107).
- [159] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. C++ In-Depth Series. Boston, Massachusetts, USA: Addison Wesley Professional, Dec. 20, 2001. ISBN: 978-0-201-72914-6. URL: <http://www.boost.org/libs/graph/doc/index.html> (visited on 01/05/2011) (cit. on pp. 124, 126).
- [160] Arvind Sridhar et al. “3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling”. In: *Proceedings of the 2010 International Conference on Computer-Aided Design (ICCAD 2010)*. (San Jose, CA, USA, Nov. 7–11, 2010). Vol. 1. IEEE. New York: ACM and IEEE Press, 2010, pp. 463–470. ISBN: 978-1-4244-8193-4. DOI: 10.1109/ICCAD.2010.5653749 (cit. on p. 7).
- [161] Greg Stein and Jim Whitehead. *WebDAV Resources*. 1999–2010. URL: <http://www.webdav.org/> (visited on 10/12/2010) (cit. on p. 28).
- [162] Horst Stöcker, ed. *Taschenbuch mathematischer Formeln und moderner Verfahren*. German. 4th ed. Thun und Frankfurt am Main: Verlag Harri Deutsch, 1999. ISBN: 3-8171-1573-3 (cit. on pp. 110, 118, 131, 140).
- [163] Bjarne Stroustrup. *Die C++ Programmiersprache*. German. Trans. from the English by Nicolai Josuttis and Achim Lörke. 4th ed. Addison-Wesley, 2000. ISBN: 3-8272-1660-X (cit. on pp. 92, 95, 100).
- [164] SystemC-AMS Study Group. *Homepage of the SystemC-AMS Study Group*. SystemC-AMS Study Group. 2002–2010. URL: <http://www.systemc-ams.org/> (visited on 10/12/2010) (cit. on pp. 14, 15, 72).
- [165] Jean U. Thoma. *Introduction to Bond Graphs and Their Applications*. Oxford: Pergamon Press, 1975 (cit. on p. 16).

- [166] Dirk Thomas, Quinn Taylor, and Tim Armes. *WebSVN—Online subversion repository browser*. 2008–2010. URL: <http://www.websvn.info/> (visited on 12/12/2010) (cit. on pp. 25, 28).
- [167] Jonathan Turkanis. *The Boost Iostreams Library*. 2004–2008. URL: <http://www.boost.org/libs/iostreams/doc/index.html> (visited on 01/13/2011) (cit. on p. 97).
- [168] Thomas Uhle and Karsten Einwich. “A SystemC AMS Extension for the Simulation of Non-linear Circuits”. In: *Proceedings of the 23 IEEE International SoC Conference (SOCC) 2010*. (Las Vegas, NV, USA, Sept. 27–29, 2010). IEEE. 2010, pp. 193–198. ISBN: 978-1-4244-6683-2. URL: <http://www.ieee-socc.org/SOCC2010/> (visited on 12/23/2010) (cit. on pp. 15, 84).
- [169] A. Vachoux, C. Grimm, and K. Einwich. “SystemC-AMS requirements, design objectives and rationale”. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE) 2003*. (Messe Munich, Germany, Mar. 3–7, 2003), pp. 388–393. URL: http://www.date-conference.com/proceedings/PAPERS/2003/DATE03/pdffiles/05a_5.pdf (visited on 01/05/2011) (cit. on p. 12).
- [170] Alain Vachoux, Christoph Grimm, and Karsten Einwich. “Extending SystemC to support mixed discrete-continuous system modeling and simulation”. In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2005*. (Kobe, Japan, May 23–26, 2005). Vol. 5. IEEE. 2005, pp. 5166–5169. ISBN: 0-7803-8834-8. DOI: 10.1109/ISCAS.2005.1465798 (cit. on pp. 12–14, 71, 125).
- [171] Geert Van der Plas, Georges Gielen, and Willy Sansen. *A Computer-Aided Design and Synthesis Environment for Analog Integrated Circuits*. Boston, Dordrecht, London: Kluwer Academic Publishers, 2002 (cit. on p. 2).
- [172] Jan J. van Dixhoorn. “Network Graphs and Bond Graphs in Engineering Modelling”. In: *Annals of Systems Research* 2 (1972), pp. 22–38 (cit. on p. 16).
- [173] Dimitri van Heesch. *Doxygen: Generate documentation from source code*. 1997–2010. URL: <http://www.doxygen.org/> (visited on 12/09/2010) (cit. on pp. 19, 25, 101).
- [174] Michel Vasilevski et al. “Modeling and Refining Heterogeneous Systems with SystemC-AMS: Application to WSN”. In: *Proceedings of the Design Automation & Test in Europe (DATE) Conference 2008*. (Munich, Germany, Mar. 10–14, 2008). EDAA. 2008. ISBN: 978-3-9810801-3-1. URL: http://www.date-conference.com/proceedings/PAPERS/2008/DATE08/PDFFILES/02_2_4.PDF (visited on 11/11/2011) (cit. on p. 15).
- [175] Todd Veldhuizen. *Using C++ Trait Classes for Scientific Computing*. Tech. rep. Waterloo, Ontario, Canada, N2L 3G1: Dept. of Systems Design Engineering, University of Waterloo, 1996-03-30. URL: <http://www.oonumerics.org/blitz/traits.html> (visited on 01/14/2011) (cit. on p. 92).
- [176] Virtual Dynamics. *Bond graph toolbox for Mathematica. The native symbolic bond graph tool to enhance your mechatronics creativity*. Virtual Dynamics. 1995–2010. URL: <http://www.virtualdynamics.fr/software.html> (visited on 01/10/2011) (cit. on p. 17).
- [177] Daniel von dem Knesebeck. “Development of a Web-Based Model Library of AMS Hardware Systems”. Diplomarbeit. Fakultät für Informatik (FIN), Postfach 4120, D-39016 Magdeburg: Otto-von-Guericke-Universität Magdeburg, July 4, 2008 (cit. on pp. 33, 34).

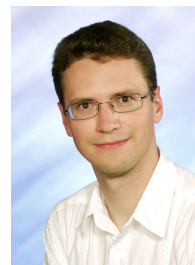
- [178] P. Wambacq et al. “Dataflow simulation of mixed-signal communication circuits using a local multirate, multicarrier signal representation”. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 49 (Nov. 2002), pp. 1554–1562. ISSN: 1057-7122. DOI: 10.1109/TCSI.2002.804579 (cit. on p. 8).
- [179] Wolfram Research. *Wolfram Mathematica: Technical Computing Software—Taking You from Idea to Solution*. Wolfram Research. 1988–2011. URL: <http://www.wolfram.com/mathematica/> (visited on 01/06/2011) (cit. on p. 17).
- [180] Tao Xu et al. “A precise SystemC-AMS model for Charge Pump Phase Lock Loop with multiphase outputs”. In: *Proceedings of the IEEE 8th International Conference on ASIC (ASICON) 2009*. (Changsha, Hunan, China, Oct. 20–23, 2009), pp. 50–53. ISBN: 978-1-4244-3868-6. DOI: 10.1109/ASICON.2009.5351608 (cit. on p. 15).
- [181] Yaseen Zaidi, Christoph Grimm, and Jan Haase. “On Mixed Abstraction, Languages, and Simulation Approach to Refinement with SystemC AMS”. In: *EURASIP Journal on Embedded Systems 2010 (2010): Design Methodologies and Innovative Architectures for Mixed-Signal Embedded Systems*. Ed. by Sergio Saponara et al., p. 13. DOI: 10.1155/2010/489365 (cit. on p. 13).
- [182] Jun Zhu, Ingo Sander, and Axel Jantsch. “HetMoC: Heterogeneous Modelling in SystemC”. In: *Proceedings of the 13th International Forum on specification & Design Languages (FDL) 2010*. (Southampton, UK, Sept. 14–16, 2010). ECSI (cit. on p. 13).
- [183] Leor Zolman. “An STL Error Message Decryptor for Visual C++”. In: *Dr. Dobb’s C/C++ Users Journal* (July 2001). URL: <http://www.ddj.com/cpp/184401416?pgno=7> (visited on 01/12/2011) (cit. on p. 95).
- [184] Leor Zolman. *STLFilt: An STL Error Message Decryptor for C++*. BD Software. 2001–2008. URL: <http://www.bdsoft.com/tools/stlfilt.html> (visited on 01/12/2011) (cit. on p. 95).

Curriculum Vitæ

Personal Details

Torsten Mähne
Avenue Edouard-Dapples 5
CH-1006 Lausanne
Switzerland

Phone: +41 (21) 6 01 37 49
Mobile: +41 (78) 9 20 83 49
E-mail: torsten.maehne@a3.epfl.ch



Date/place of birth: 14/12/1979 in Lutherstadt Eisleben, Germany
Nationality: German; Gender: Male; Marital status: Single

Education

- 03/2005–02/2011 **Ph.D. degree in microsystems and microelectronics**, *Laboratoire de Systèmes Microélectroniques (LSM), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.*

Research on efficient modelling and simulation methodologies for mixed-signal systems-on-a-chip.
- 05/2004–10/2004 **“Diplomarbeit” (Master’s thesis)**, *research department FV/FLD, Robert Bosch GmbH, Gerlingen-Schillerhöhe, Germany. “Ordnungsreduktionsverfahren zur automatischen Generierung von Systemmodellen bei mikroelektromechanischen Systemen”.*

Development of a reduced order modelling methodology for electrostatically actuated surface micromechanical systems (e.g., yaw rate sensors).
- 06/2003–08/2003 **“Studienarbeit” (semester project)**, *Institut für Sensor- und Mikrosysteme (IMOS), Otto-von-Guericke-Universität, Magdeburg, Germany. “Erstellen eines Differenzdruckgebers anhand des Prandtlschen Staurohrs”.*

Design and simulation of a piezo-resistive differential pressure sensor suited for a pitot-static tube with the help of a developed parametrisable finite elements model in ANSYS.
- 10/1999–12/2004 **“Diplom-Ingenieur (Dipl.-Ing.)” (comp. to Master of Science) in information technology with a major in microsystems and microelectronics**, *Otto-von-Guericke-Universität, Magdeburg, Germany.*

Special emphasis on semiconductor technology, integrated circuits, sensors, and microsystem technology. Passing **with distinction** the “Diplomprüfung” (final examination). **Laureate of the “Fakultätspreis 2005” for the best alumnus of the faculty of electrical engineering.**

08/1991–07/1998 **Secondary school education, Martin-Luther-Gymnasium, L. Eisleben, Germany.**
Granting of “Allgemeine Hochschulreife” (university entrance qualification) with mark 1.0 (excellent).

Working Experience

03/2005–02/2011 **Research assistant, Laboratoire de Systèmes Microélectroniques (LSM), École Polytechnique Fédérale de Lausanne (EPFL)** in partial collaboration with CSEM SA at Neuchâtel, Switzerland.

- Development of support for dimensional analysis and bond graphs in SystemC-AMS to improve its modelling capabilities for multiphysical systems.
- Contributions (proposals, reviews, examples) to the standardisation of AMS extensions for SystemC as *member of the OSCI AMSWG* since 07/2007.
- Development of a modelling methodology and behavioural model library for RF transceivers.
- Development of a web-based platform “ModelLib” for collecting behavioural models and supporting the design of AMS systems.
- Supervision of three Master’s student projects.
- Teaching assistance for courses on hardware systems modelling (7 terms), semi-custom digital design flows (5 terms), and C++ programming (1 term) in English and French language.

10/2003–10/2004 **Work placement and “Diplomarbeit” (final thesis), research department FV/FLD, Robert Bosch GmbH, Gerlingen-Schillerhöhe, Germany.**

Development of a reduced-order modelling methodology for electrostatically actuated surface micromechanical systems (e.g., yaw rate sensors) allowing to augment their mechanical FE models in ANSYS with electrostatic transducer elements and to extract VDHL-AMS models for system-level simulations.

08/2002–08/2003 **Research assistant and semester project, Institut für Sensor- und Mikrosysteme (IMOS), Otto-von-Guericke-Universität, Magdeburg, Germany.**

- Development of a parametrisable FE model in ANSYS to design a piezo-resistive differential pressure sensor.
- Setting up a PVD System and a wafer on adhesive film mounter.
- Operating the wafer dicing saw.

04/2000–07/2002 **Research assistant, Institut für Technische und Betriebliche Informationssysteme (ITI), Otto-von-Guericke-Universität, Magdeburg, Germany.**

- Co-authoring the book and developing examples for the lecture “Grundlagen der Informatik für Ingenieure”.
- Converting a casting technology database from Informix HyperScript Tools to Oracle 8.

Publications

Books

- [1] Georg Paul et al. *Grundlagen der Informatik für Ingenieure – Eine Einführung mit C/C++*. German. Stuttgart, Leipzig: B.G. Teubner Verlag, 2003. ISBN: 3-519-00428-3. URL: http://www.witi.cs.uni-magdeburg.de/iti_ti/IngInf/.

Book Chapters

- [2] Torsten Maehne et al. “A VHDL-AMS Modeling Methodology for Top-Down/Bottom-Up Design of RF Systems”. In: *Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's – Selected Contributions on Specification, Design, and Verification from FDL 2009*. Ed. by Dominique Borrione. 1st ed. Vol. 63. Lecture Notes in Electrical Engineering (LNEE). Springer, Aug. 2010. ISBN: 978-90-481-9303-5.
- [3] Torsten Mähne and Alain Vachoux. “ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems”. In: *Advances in Design and Specification Languages for Embedded Systems. Selected Contributions from FDL'06*. Ed. by Sorin A. Huss. ChDL. Dordrecht, The Netherlands: Springer Verlag, July 2007. Chap. 4, pp. 53–72. ISBN: 978-1-4020-6147-9.
- [4] Torsten Mähne et al. “Creating Virtual Prototypes of Complex MEMS Transducers Using Reduced-Order Modelling Methods and VHDL-AMS”. In: *Applications of Specification and Design Languages for SoCs. Selected papers from FDL'05*. ChDL. Springer, 2006, pp. 135–153. ISBN: 1-4020-4997-8.

Conference Papers

- [5] Torsten Maehne et al. “A VHDL-AMS Modeling Methodology for Top-Down/Bottom-Up Design of RF Systems”. In: *Proceedings of the 12th International Forum on specification & Design Languages (FDL) 2009*. (Sophia Antipolis, France, Sept. 22–24, 2009). ECSI. 2009. URL: <http://infoscience.epfl.ch/record/138641> (visited on 10/12/2010).
- [6] Torsten Maehne and Alain Vachoux. “Supporting Dimensional Analysis in SystemC-AMS”. In: *Proceedings of the 2009 IEEE International Behavioral Modeling and Simulation (BMAS) Workshop*. (Doubletree Hotel, San Jose, California, USA, Sept. 17–18, 2009). IEEE. 2009, pp. 108–113. DOI: 10.1109/BMAS.2009.5338878. URL: <http://infoscience.epfl.ch/record/140571> (visited on 03/17/2010).
- [7] Torsten Mähne, Alain Vachoux, and Yusuf Leblebici. “Support pour l'analyse dimensionnelle en SystemC-AMS”. French. In: *Résumé des présentations de l'École d'hiver Francophone sur les Technologies de Conception des systèmes embarqués Hétérogènes (FETCH) 2009*. (Hôtel Préalpina, Chexbres, Suisse, Jan. 12–14, 2009). École Polytechnique Fédérale de Lausanne (EPFL). 2009, pp. 50–51. URL: <http://infoscience.epfl.ch/record/131186> (visited on 10/12/2010).

- [8] Alain Vachoux and Torsten Maehne. “SystemC-Based Modeling of Embedded Heterogeneous Systems”. In: *Proceedings of the Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference (NEWCAS-TAISA) 2008*. (Montréal, Québec, Canada, June 22–25, 2008). IEEE. 2008, pp. 277–280. ISBN: 978-1-4244-2331-6. DOI: 10.1109/NEWCAS.2008.4606375.
- [9] Torsten Maehne, Alain Vachoux, and Yusuf Leblebici. “Development of a Bond Graph Based Model of Computation for SystemC-AMS”. In: *Proceedings of the 4th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME) 2008*. (Boğaziçi Üniversitesi, İstanbul, Turkey, June 22–25, 2008). IEEE. 2008. ISBN: 978-1-4244-1983-8. DOI: 10.1109/RME.2008.4595729. URL: <http://infoscience.epfl.ch/record/121405> (visited on 10/12/2010).
- [10] Torsten Mähne, Alain Vachoux, and Yusuf Leblebici. “Développement d’un modèle de calcul Bond Graph pour SystemC-AMS”. French. In: *École d’hiver Francophone sur les Technologies de Conception des systèmes embarqués Hétérogènes (FETCH) 2008 : Présentations des Doctorants*. (Montebello, Canada, Jan. 7–9, 2008). École Polytechnique Montréal. 2008. URL: <http://infoscience.epfl.ch/record/116443> (visited on 10/12/2010).
- [11] Torsten Maehne and Alain Vachoux. “Proposal for a Bond Graph Based Model of Computation in SystemC-AMS”. In: *Proceedings of the 10th Forum on specification & Design Languages (FDL) 2007*. (Barcelona, Sept. 18–20, 2007). ECSI. 2007, pp. 25–31. ISBN: 978-2-9530504-0-0. URL: <http://infoscience.epfl.ch/record/108824> (visited on 10/11/2010).
- [12] Torsten Mähne, Alain Vachoux, and Yusuf Leblebici. “Fostering the Reuse and Collaborative Development of Models in the AMS SoC Design Process”. In: *Proceedings of the 2007 Ph.D. Research in Microelectronics and Electronics (PRIME) Conference*. (Bordeaux, France, July 2–5, 2007). IEEE. 2007, pp. 285–288. ISBN: 1-4244-1000-2. DOI: 10.1109/RME.2007.4401868. URL: <http://prime07.ixl.fr/> (visited on 10/11/2010).
- [13] Torsten Maehne, Alain Vachoux, and Yusuf Leblebici. “Proposal to Extend SystemC-AMS with a Bond Graph Based Model of Computation”. In: *Proceedings of the C/C++-Based Modelling of Embedded Mixed-Signal Systems Workshop 2007*. (Dresden, June 25–26, 2007). Fraunhofer Institut für Integrierte Schaltungen, Außenstelle Entwurfsautomatisierung, 2007, pp. 41–53. URL: http://www.eas.iis.fraunhofer.de/events/workshops/2007/cmемss/index_de.html.
- [14] Torsten Mähne, Alain Vachoux, and Yusuf Leblebici. “Méthodes de modélisation efficaces pour la conception de systèmes monopuces hétérogènes”. French. In: *Ecole d’hiver Francophone sur les Technologies de Conception des systèmes embarqués Hétérogènes (FETCH) 2007 : Programme et Résumés de thèses*. (Villard-de-Lans, France, Jan. 10–12, 2007). Ed. by Ahmed-Amine Jerraya. EPFL et al. 2007. URL: <http://leom.ec-lyon.fr/fetch/> (visited on 10/11/2010).
- [15] Torsten Mähne and Alain Vachoux. “ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems”. In: *Proceedings of the 9th International Forum on Specification and Design Languages (FDL) 2006*. (Darmstadt University, Germany, Sept. 19–22, 2006). ECSI. 2006, pp. 91–97. ISBN: 978-3-00-019710-9. URL: <http://infoscience.epfl.ch/search.py?recid=88137> (visited on 10/11/2010).
- [16] Torsten Mähne et al. “Creating Virtual Prototypes of Complex Micro-Electro-Mechanical Transducers Using Reduced-Order Modelling Methods and VHDL-AMS”. In: *Proceedings of the 8th International Forum on specification and Design Languages (FDL) 2005*. (École Polytech-

nique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, Sept. 27–30, 2005). ECSI. 2005, pp. 209–221. URL: <http://infoscience.epfl.ch/search.py?recid=62708> (visited on 10/11/2010).

Other Publications

- [17] F. Giroud et al. “A VHDL-AMS Modeling Methodology for Top-Down/Bottom-Up Design of RF Systems”. In: *CSEM Scientific and Technical Report 2009*. Rue Jaquet-Droz 1, P.O. Box, CH-2002 Neuchâtel, Switzerland: Centre Suisse d’Électronique et de Microtechnique SA (CSEM), 2010, p. 34. URL: <http://www.csem.ch/docs/show.aspx?id=12660> (visited on 10/20/2010).
- [18] Martin Barnasconi et al. *SystemC AMS extensions User’s Guide*. Open SystemC Initiative (OSCI). Mar. 8, 2010. URL: http://www.systemc.org/members/download_files/check_file?agreement=ams_ext_10std (visited on 10/12/2010).
- [19] Torsten Mähne. “Ordnungsreduktionsverfahren zur automatischen Generierung von Systemmodellen bei mikroelektromechanischen Systemen”. German. Diplomarbeit. Fakultät für Elektrotechnik und Informationstechnik (FEIT), Postfach 4120, D-39016 Magdeburg, Germany: Otto-von-Guericke-Universität Magdeburg, Nov. 1, 2004.
- [20] Torsten Mähne. “Erstellen eines Differenzdruckgebers anhand des PRANDTLschen Staurohrs”. German. Studienarbeit. Fakultät für Elektrotechnik und Informationstechnik (FEIT), Postfach 4120, D-39016 Magdeburg, Germany: Otto-von-Guericke-Universität Magdeburg, Sept. 2003.

Invited Presentations

- [21] Torsten Maehne and Alain Vachoux. *Multi-domain SoC Modeling With SystemC-AMS: Employing Dimensional Analysis and Bond Graphs*. Invited presentation in the scope of SoC seminar at University of British Columbia, Vancouver, British Columbia, Canada. Sept. 15, 2009.
- [22] Torsten Maehne and Alain Vachoux. *Bond Graph as a Model of Computation in SystemC-AMS*. Invited presentation at the Hardware Verification Group of Concordia University, Montréal, Québec, Canada. Jan. 8, 2008.
- [23] Alain Vachoux and Torsten Maehne. *SystemC-Based Modeling of Embedded Heterogeneous (AMS) Systems*. Invited presentation at the Hardware Verification Group of Concordia University, Montréal, Québec, Canada. June 26, 2008.

